ADAPTIVE QUALITY OF SERVICE ENGINE WITH DYNAMIC QUEUE CONTROL

THESIS

James Haught

AFIT/GCS/ENG/11-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/11-03

ADAPTIVE QUALITY OF SERVICE ENGINE WITH DYNAMIC QUEUE CONTROL

THESIS

Presented to the Faculty

Department of Electrical & Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

James Haught

March 2011

AFIT/GCS/ENG/11-03

ADAPTATIVE QUALITY OF SERVICE ENGINE WITH DYNAMIC QUEUE CONTROL

James Haught

Approved:

/signed/	10 Mar 2011
Dr. Kenneth M. Hopkinson (Chairman)	Date
/signed/	10 Mar 2011
Lt. Col. Brett J. Borghetti, PhD (Member)	Date
/signed/	10 Mar 2011
Maj. Jeffrey M. Hemmes, PhD (Member)	Date

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Hopkinson, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. I would also like to thank the following people; Dr. Shay Capehart with the assistance of developing the Kalman Rating method, Michael Dop for porting the Kalman filter code from OPNET to ns2 and assistance in the understanding of ns2, Joe Wilhelm and Matthew Ludwig for the assistance with the MATLAB code used in some of the analysis work in this thesis, Jose Fadul for the assistance with the ns2 code used to dynamically change routing tables in ns2, Alexander Stirling for the editing of the Winter Simulation conference paper that was published with this thesis, and Mark Duncan for skillful assistance in methodology and network performance modeling concepts.

James Haught

Table of Contents

Page

List of Figures

Page

List of Tables

Page

Abstract

While the current routing and congestion control algorithms in use today are often sufficient for networks with relatively static topology, these algorithms may not be sufficient for military networks where a certain level of quality of service (QoS) needs to be achieved to complete a mission. Current networking technology limits a network's ability to adapt to changes and interactions in the network, often resulting in sub-optimal performance. This research investigates the use of queue size predictions to create a network controller to optimize computer networks. These queue size predictions are made possible through the use of Kalman filters to detect network congestion. The premise is that intelligent agents can use such predictions to form context-aware, cognitive processes for managing communication in mobile networks. The network controller designed and implement in this thesis will take in the current and predicted network conditions and make intelligent choices to optimize the network.

ADAPTATIVE QUALITY OF SERVICE ENGINE WITH DYNAMIC QUEUE CONTROL

I. Introduction

While the current routing and congestion control algorithms in use today are often sufficient for networks with relatively static topology, these algorithms may not be sufficient for military networks where a certain level of quality of service (QoS) needs to be achieved to complete a mission. Current networking technology limits a network's ability to adapt to changes and interactions in the network, often resulting in sub-optimal performance. Limited in state, scope, and response mechanisms, the network elements (consisting of nodes, protocol layers, policies, and behaviors) are unable to make intelligent adaptations to meet network-wide goals. Communication of network state information is stifled by the layered protocol architecture, making individual elements unaware of the network conditions experienced by other elements. Any response that an element may make to network stimuli can only be made inside of its limited scope. The adaptations that are performed are typically reactive, taking place only after a problem has occurred.

There exists across the field of computer networking the need to achieve network-level objectives in the face of increasing network complexity. Particularly in wireless networks, there has been a trend towards increasingly heterogeneous and dynamic environments. Past research has been investigating a radical new paradigm where the cognitive network using distributed intelligent agents, to autonomously and dynamically achieve complex network level objectives in a wireless network. A network with

distributed intelligence utilizes cognition in the network which is defined loosely as the ability to perceive current conditions. The intelligent agents in the network then plan and decide how to act on the current perceived network conditions. The intelligent agents learn to make decisions that take end-to-end goals into account. The agents in the cognitive network cooperate in a peer-to-peer manner to create an intelligent distributed system. Making this vision a reality requires advances in intelligent network optimization, wide-area network monitoring, and distributed routing. In particular, this thesis concentrates on a wide-area network monitor and prediction system, which will form as an important input for this type of distributed agent framework.

To provide automated decision support for network level object; this research investigates the broader problem of linking users to the networking infrastructure that they operate on. The main idea is that by giving users a better picture of the status and overall capabilities of the network, the cognitive (i.e. intelligent) processes in the distributed network will be given clearer, more accurate insight into the mission objectives. An illustration of this is given in Figure 1.1, showing how current conditions are combined with user preferences and network objectives to determine the best network configuration. This may not easily be applied directly to large chaotic networks, like the Internet, but this system could be applied to many special-purpose networks for corporations, critical infrastructure management, and military command and control. These networks could be better predicted and controlled than they are today with the right inputs and distributed management framework.

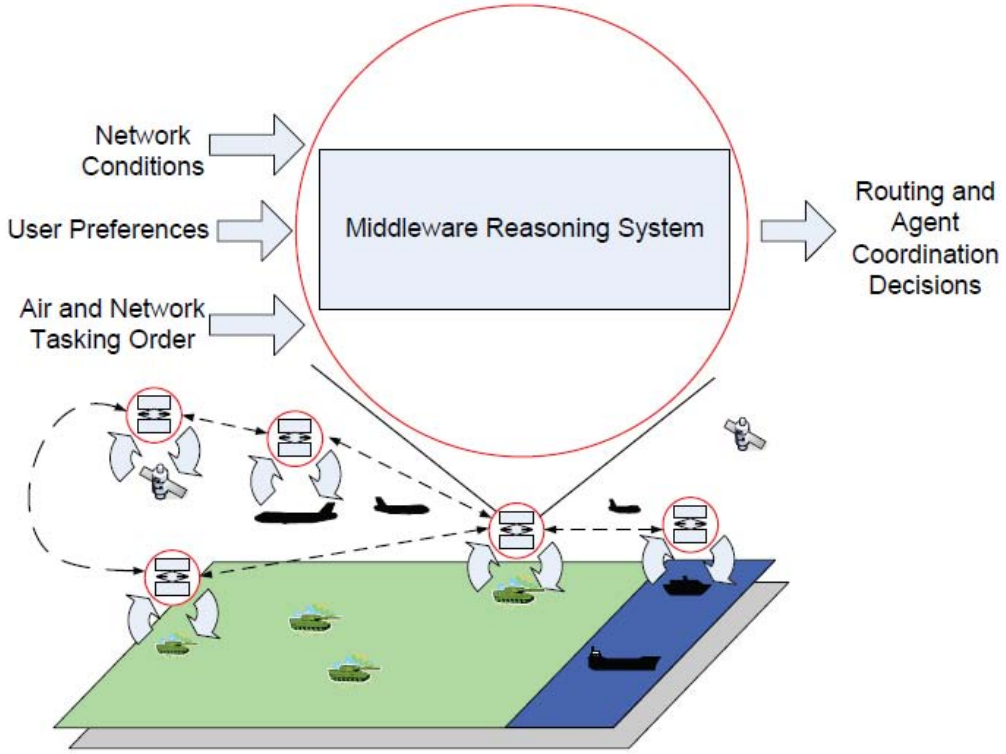

Figure 1.1: An illustration of mobile network using middleware to enhance the reliability and quality of service (QoS) properties of the system.

Major elements of this vision have been created over the past few years. Air Force tactical environments are typically planned at least a day ahead, in a document called an Air Tasking Order [1], such information could be incorporated into what we call a Network Tasking Order [1] in order to feed medium to long-term information into our agent-based network management framework. Previous research has shown that such information could enable more optimal network outcomes if it were available [1-3]. Past work has also demonstrated the ability of an agent-based framework to optimize network behavior using long-term, mid-term, and short-term estimates of network behavior [2].

This thesis presents one approach for making medium-term estimates of network conditions. This is made possible through the use of Kalman filters. An application is developed in this thesis to use such estimates to manage the network traffic to improve its priority and Quality of Service (QoS) characteristics. The idea is to attempt to maximize the mission impact of the network traffic based on current conditions. Experimental results illustrate the promise of this approach.

II. Literature Review

This chapter presents research related to the use of intelligent agents within a network environment. This chapter also explains the Kalman filter algorithm employed in this thesis. Notable related research presented includes the intelligent agent work performed at Air Force Institute of Technology (AFIT) through the Network Weatherman research [3]. This literature review also describes work related to network optimization and control theory.

Network Optimization

In the areas of applied mathematics, network science, and graph theory, the topic of *network theory* concerns itself with the study of graphs as a representation of asymmetric relations between discrete objects [4]. Network theory problems that involve finding optimal ways of performing a task are studied under the name *combinatorial optimization*, which when pertaining to networks is known as *network optimization.* The goal of network optimization is to solve problems where the set of feasible solutions is discrete or can be reduced to be discrete. Simply stated, network optimization attempts to find the best possible solution to the network routing and/or topology. Some examples of network optimization include network flows, the shortest path problem, transportation problems, matching problems, and routing problems. The primary focus of this section deals with network flow problems.

The objective of a network flow is to move some quantity of units (electricity, water, time, etc.), in our case data packets, from one point to another through an existing network as efficiently as possible [5]. In graph theory, a flow network is a directed graph where each edge has a capacity and is designated a flow value. The amount of flow on an edge cannot exceed the capacity of that edge. In Operations Research, a directed graph is called a network, where the vertices are known as *nodes* and edges are known as *arcs*. A flow must satisfy the restriction that the amount of flow into a node equals the flow out of it, except when that node is a *source*. The source can have infinite outbound flow and the sink can have infinite inbound flow.

A network flow model is represented as a directed network $G = (N, A)$ defined by a set N of nodes and a set A of directed arcs [6]. Each arc (i, j) ϵ A has an associated cost C_{ij} that denotes the cost per unit flow on the arc. The flow cost varies linearly with the amount of flow [6]. Also associated with (i, j) is an upper bound u_{ij} on the arc capacity that denotes the maximum flow on (i, j) and a lower bound l_{ij} that denotes the minimum flow on the arc [6]. Each node $i \epsilon N$ is characterized by an integer $b(i)$ representing supply/demand [6]. The decision variables in the network flow problem are the arc flows (i, j) and are represented by x_{ij} [6].

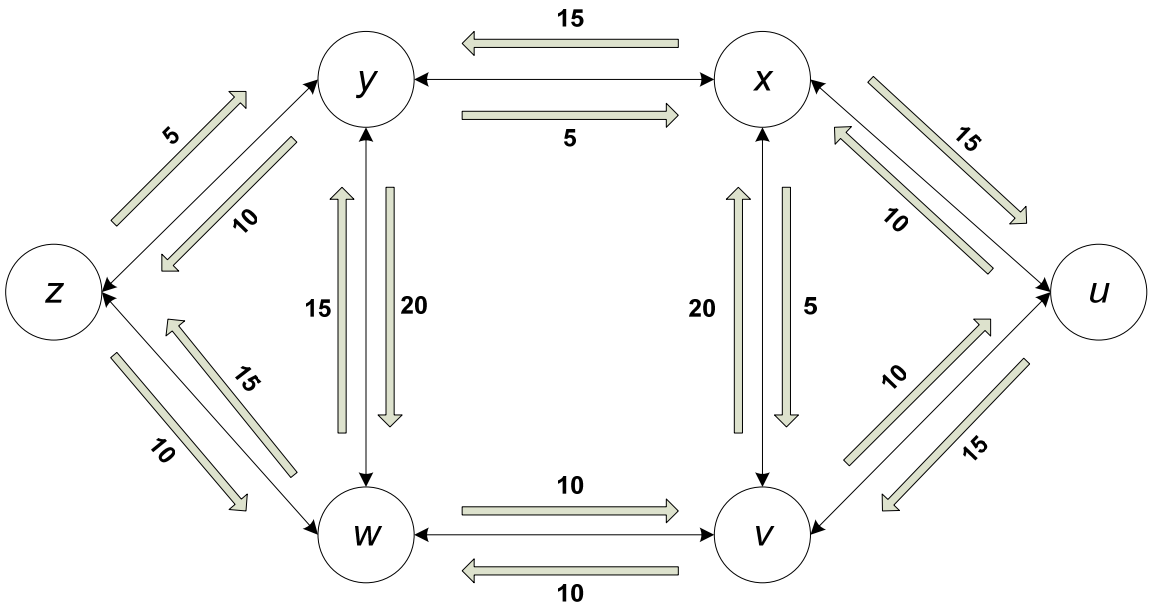

Figure 2.1: An example of a directed graph showing the flows of the network.

The network flow problem can be formulated as:

Minimize $\sum_{(i,j)\epsilon A} C_{ij}x_{ij}$ subject to constraints:

1) $\sum_{j:(i,j)\epsilon A} x_{ij} - \sum_{j:(j,i)\epsilon A} u_{ji}x_{ji} = b(i) \quad \forall i \epsilon N;$
2) $l_{ij} \leq x_{ij} \leq u_{ij}.$

Constraint (1) is called the *mass balance constraint* which states that the difference between the flow into node j and the flow out of node j must equal its supply/demand of flow [6]. However, in a circulation version of a network flow problem, such as the one used here, $b(i) = 0 \; \forall \; i \epsilon N$ and therefore no flow can accumulate at any node $i \epsilon N$ [6]. Constraint (2) is called the *flow constraint* which states that the flow along any arc (i, j) must lie between its lower and upper capacity bounds. In any feasible solution, every arc flow x_{ij} must satisfy both these constraints [6].

The simplest and most common problem using flow networks is finding what is known as *maximum flow*, which provides the largest possible total flow from the source

to the sink in a given network [7]. Part of the overall goal in this research is to optimize the network's information flow so that critical network flows related to the mission are given priority.

Kalman Filter

Humans have been filtering and sorting things throughout our entire past; we can filter water with our hands to skim dirt and leaves from the surface of the water. We selectively filter out small noises (traffic, appliances, etc.) by focusing on important sounds, like the voice of the person to whom we're speaking with [8]. In the field of engineering, filtering is desirable for radio communication signals that are often corrupted with noise. A good filtering algorithm can remove this noise from electromagnetic signals while still retaining its useful information [8].

Weather forecasts are infrequently accurate as they are based on average predictions from previous data. The accuracy of a weather forecast is derived from the uncertainty of the measurements of data. In statistics, variance is the most common measure of uncertainty. The Kalman filter is used to calculate forecasts and forecast variances for weather models [9].

Although Rudolf E. Kalman developed Kalman filtering to solve a spacecraft navigation problem for the Apollo space program in 1960, it has roots far back as Karl Gauss in 1795 with his method of least squares [8]. A Kalman filter is a designed recursive solution to the discrete-data linear filtering problem. In the theory of stochastic processes, the discrete-data linear filtering problem involves formulating a best estimate

of the true value of the system when given a set of potentially noisy observations from that system [8]. The Kalman filter has been the subject of extensive research and a wide range of engineering applications from radar to computer vision, and is an important topic in control theory. The *Kalman filter* is the best known Least Mean-Square (LMS) algorithm to optimally estimate the unknown state of a dynamic system from a series of incomplete and noisy measurements [10]. The Kalman filter is a set of mathematical equations that provides the means to estimate the state of a process. The main benefit of the Kalman filter is that it supports estimations of past, present, and even future states of dynamic systems.

The Kalman filter is a recursive estimator that is constantly in a one of two distinct phases, *predict* and *update*, which are illustrated in Figure 2.2. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state with no history of observations or estimates required. The predict phase uses the Kalman filter's state estimate from the previous time to produce an estimate of the state at the current time, this state is also known as the *priori* state because it is an estimate of the state at the current time step. The update phase combines *priori* prediction with the current observed information into a refined state estimate called the *posteriori* state estimate.

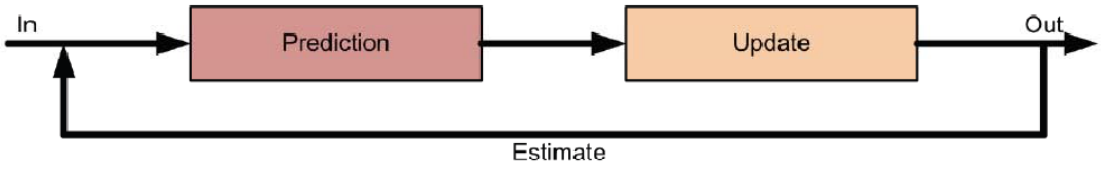

Figure 2.2: Kalman filter cycle [11].

The Kalman filter is modeled in the following equation:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \quad (2.1)\ [12]$$

Where

- F_k is the state transition model which is applied to the previous state of the model.
- B_k is the control-input model which is applied to the control vector u_k.
- w_k is the process noise which is to be from a zero mean normal distribution with covariance Q_k in the following equation.

$$w_k \sim N(0, Q_k) \qquad (2.2)\ [12]$$

At time *k* a measurement z_k of the true state x_k is made according to

$$z_k = H_k x_k + v_k \qquad (2.3)\ [12]$$

Where H_k is the observation model that maps the true state space into the observed space and v_k is the observation noise which is assumed to be zero mean Gaussian white noise and covariance R_k in the following equation:

$$v_k \sim N(0, R_k) \qquad (2.4)\ [12]$$

The state of the Kalman filter is represented by two variables:

- $\hat{x}_{k|k}$ is a *posteriori* state estimate at time *k* given observations up to and including at time *k* [12].

- $P_{k|k}$ is a *posteriori* error covariance matrix, which is a measure of estimated accuracy of the state estimate [12].

The notion $\hat{x}_{n|m}$ represents the estimate of *x* at time *n* given observations up to, and including time *m* [12].

The Kalman filter uses a form of feedback control that estimates the process state at some time and obtains feedback in the form of measurements [12]. The two types of equations the Kalman filter uses are the time update equations and measurement update equations [12]. The time update equations project the future of the current state while error covariance attempts to obtain estimates for the next time step [12]. These equations are modeled and summarized in Figure 2.3. Kalman filters have been used in various areas of research including network research, which is covered in the next section.

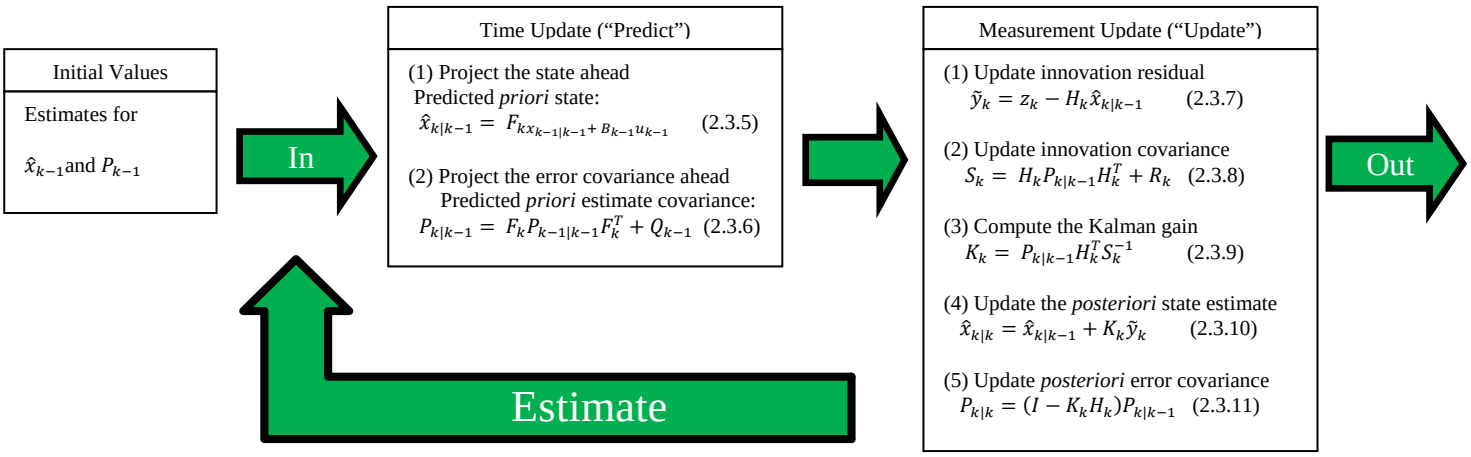

Figure 2.3: A complete overview flow chart of the Kalman filter process [13].

Kalman Filters in Network Research

Kalman filters have been used in various types of general network research, and more specifically in wireless networking. In wireless networks, there is an overhead cost for each packet of data sent through the network. If the packet size is too large then the chances of that packet being corrupted along its path to the destination is increased. If the packet is too small, the result is an inefficient overhead cost. A Kalman filter has been successfully used to estimate optimal packet size for a wireless network and is proven more effective than the moving average method [14].

The high volume of heterogeneous traffic handled by an Asynchronous Transfer Mode (ATM) network produces a need for effective flow and congestion control algorithms. A solution to this problem is to predict the future state of traffic far enough in advance to avoid congestion while efficiently utilizing the available network capacity. The estimation of the network traffic model is based on the fact that the underlying traffic model is a Markov chain. A Markov chain is a random process where all information about the future is contained in the present state. It is easier to think of a Markov chain when relating it to random events in a game like Monopoly, whose moves are determined entirely by dice. This process differs from card games, such as blackjack, where the cards represent memory of past moves. Kalman filtering has been successfully used to obtain estimates of source activity from measurements of traffic on a link and can also predict pending congestion for time intervals comparable to the round trip delay of the link [15]. The next section covers how Kalman filters have been used for predicting the future conditions of a network.

Network Forecasting

As network technology advances, the resulting improvements in network communication speed have made it possible to design and implement control algorithms to optimize overall performance. These network control algorithms use interconnected but separate computer systems as a high-performance computational platform in parallel applications in a method called *metacomputing.* "The *Network Weather Service* [NWS] is a generalizable and extensible facility designed to provide dynamic resource performance forecasts in metacomputing environments" [16]. The NWS uses a mean-based, median-based, and autoregressive methods to cast predictions of performance deliverable to the application from the available metacomputing resources.

Dynamic schedulers have been able to automatically reallocate network resources to provide better performance for applications [17]. When network performance is fluctuating, timeout determination is a crucial factor to both application and service performance. Premature timeouts cause needless failures and/or extra overhead, while overly long timeouts result in degraded messaging performance as communication is delayed unnecessarily. Based on NWS, *Adaptive Timeout Discovery* is a system that uses nonparametric statistical forecasts of request-response times to automatically determine message timeouts by choosing a timeout based on predicted network performance [18]. One of the advantages of Adaptive Timeout Discovery is that dynamic timeout discovery based on NWS conditions are more reliable and predictable performance than static methods. This allows the ability to learn network conditions and respond to actual

changes in the network conditions and can outperform TCP/IP congestion control algorithms [18].

Nathan Stuckey implemented the idea of the *Network Weatherman* (NWM) in OPNET, a network simulator [3]. NWM is an extended Kalman filter that will predict the past, present, and future states of the network queue. These predictions allow network control algorithms to optimally control the network and thus optimizing metrics of interest such as delay or throughput. Unlike the NWS, the NWM uses a Kalman filter to estimate the network state instead of a collection of data to create a mean or median based prediction; therefore the NWM doesn't need a history of past data to make its predictions and only requires the current state of the network. In Figure 2.4, a Kalman filter is placed in-between two routers which was simulated in OPNET.

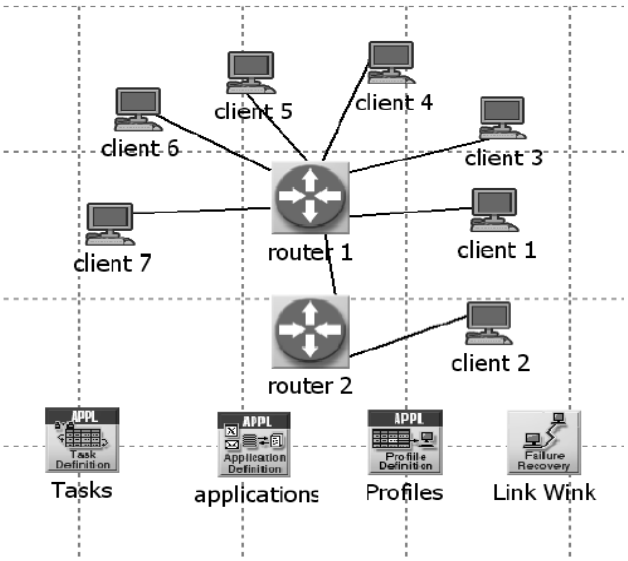

Figure 2.4: A Kalman filter placed between router 1 and router 2 in an OPNET simulation [11].

The NWM's Kalman filter can be used to predict the network's queue size [3]. Figure 2.6 displays a queue size vs. time graph of a Kalman filter that was able to predict the actual queue size ten seconds into the future from the simulation in Figure 2.5. The Kalman filter implemented by Stuckey computes the packet arrival rate using the Marcum Q-Function which is used in solving the expected value of the queue size. The behavior of the Kalman filter is represented by two states, packet arrival rate and packet service rate which are traditionally represented by λ_n and μ_n for the integer queue size n. The advantage of having an estimated queue size is that a network can use this information to optimize itself. One idea is to send specialized packets to all senders from the host so that packets can optimize their routes.

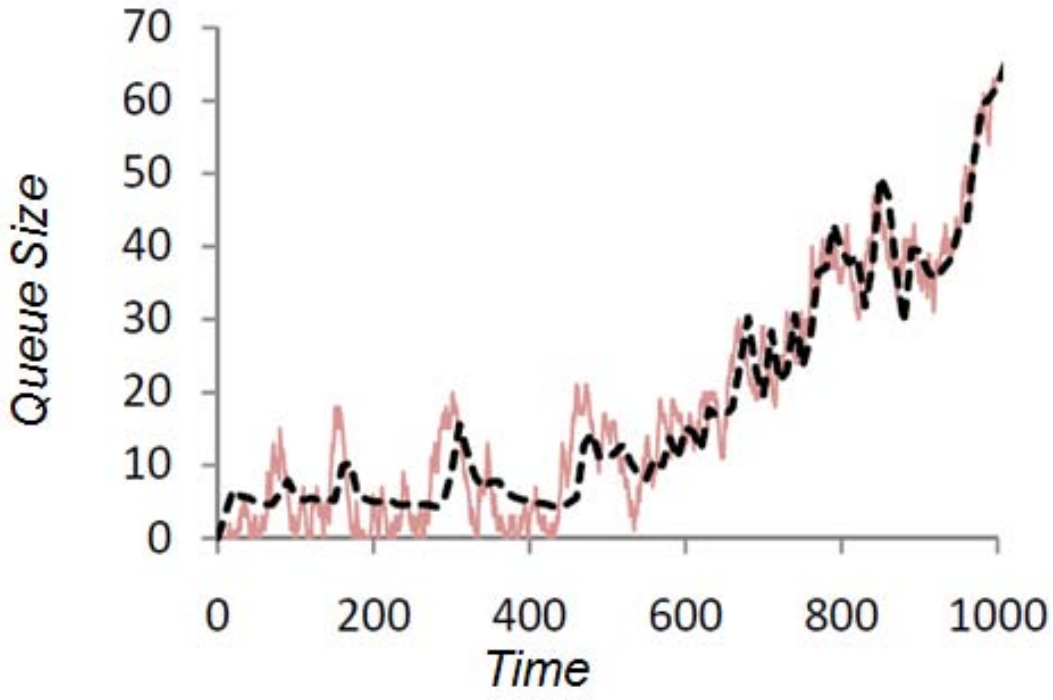

Figure 2.5: Actual queue size versus 10 second future queue size predictions of a Kalman filter placed between two nodes [11].

III. Methodology

This chapter is about the design and implementation of a Kalman filter predicting system that reacts to current and predicted network conditions. The first part of this chapter explains a validation method of evaluating the performance of Kalman filter predictions. With the validation of the Kalman filter predictions, it is possible to use these predictions to improve network conditions. The second part of this chapter describes the design and implementation of an application that reacts to these predictions called the Dynamic Route Queue Controller (DRQC).

NS2

All of the simulations that are developed and implemented in this thesis will be use Network Simulator 2 (NS2). NS2 is an event driven simulation tool that is useful in studying communication networks. Since its birth in 1989, NS2 has grown constant popularity in the research community [19]. C++ and the Object-oriented Tool Command Language (OTcl) are the two main programming languages in NS2. The C++ code defines the internal mechanism of NS2, while OTcl sets up the simulation by calling the objects and scheduled discrete events.

As seen in Figure 3.1, the NS2 engine begins by reading in a script that was coded in Tcl. Through the Tcl script, the NS2 engine creates links, nodes, network flows, and other network objects. Once these network objects are created, the simulation is executed. The Kalman filter code was added as an extension of this engine. At every prediction time, the NS2 engine sends the current queue size and current time to the

Kalman filter modular. The Kalman filter module returns the predicted queue size at the next time step. The Kalman filter records the actual queue sizes and the predicted queue sizes in a data file for analysis of how well the Kalman filter is performing. The NS2 engine makes a trace file that contains all the actions executed in the network and a Network Animator (NAM) file that is used for visualization of the network in action.

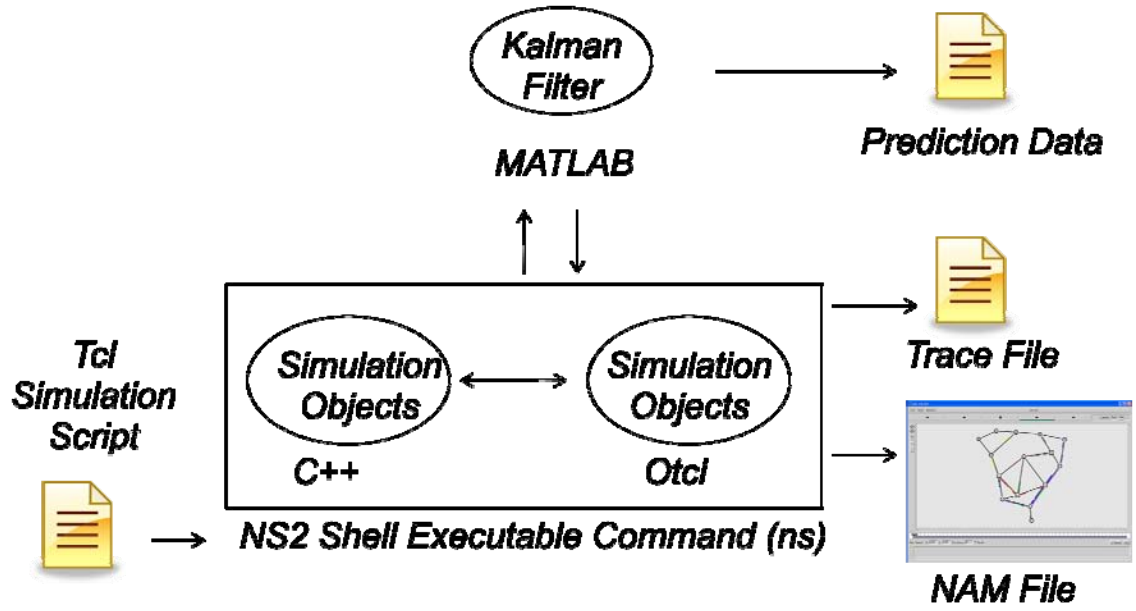

Figure 3.1: Overview chart of NS2.

Kalman Filter Validation

In order to design and implement a network controller that reacts to Kalman filter predictions, there is a need to know how accurate those predictions are in a realistic network environment. To validate the accuracy of NWM, there needs to be a comparison of future predictions against the current state of the queue at the predicted time. In previous research, only 2 nodes with multiple types of network flows were simulated in OPNET. The goal of this section is to design the methodology for validating the Kalman filter predictions in a realistic network. To make a more realistic network there will be more nodes, links, and network flows which will be used while the Kalman filters are making predictions. The purpose of this is to show what a network controller would expect from the predictions made from the Kalman filters.

The main method of validating the performance of Kalman filters in a more realistic network is by placing multiple Kalman filters in a large network. In this network, many network flows are simulated to emulate a real network. The goal is to receive similar results from the previous research. This will be useful to the application that reacts to these predictions, as explained later in this chapter. In order to see if the results are similar to previous research, methods for measuring the performance of Kalman filters are used. In addition to using the previous research's methods of measuring Kalman filter performance, another method was developed which is explained in the next section. These methods are implemented into a network simulation, which is explained in Chapter IV.

Kalman Rating

In order to measure how accurate the Kalman filters are performing, there needs to be a method of rating predictions. In previous research, the mean absolute percent error (MAPE) method was used to assign a value to how well the Kalman filter performed. MAPE takes the percent error at each prediction, and then averages all the percent error values to get the overall MAPE value. The problem with this method, though, is that the MAPE value typically doesn't reflect the majority of the predictions in a simulation. For example, in Table 3.1, the MAPE is 19.7% while 8 out of the 10 of the predictions do not reflect that value. In the vision of an agent-based framework that was discussed in Chapter I, it would be more valuable to know what percentages of the predictions are good enough to be used by defining each prediction that is within a percent error tolerance as a successful prediction.

TABLE 3.1: Example of MAPE and Kalman Rating.

Time step	1	2	3	4	5	6	7	8	9	10	AVG
Actual	15	5	20	105	55	30	30	35	45	45	
Predicted	14	6	18	79	130	30	33	36	42	45	
Percent Error	6	0	10	25	136	0	10	3	7	0	**19.7%**
Success (<10%)	1	1	1	0	0	1	1	1	1	1	**80%**

The Kalman Rating is the minimum percent error tolerance where at least a certain percentage of the predictions reflect success. An example of a Kalman rating is depicted in Table 1 where the certain percentage is 80%. The percent error tolerance and Kalman rating in Table 1's case is 10%. These percent error tolerance can be adjusted to match the agent-based framework's need to know of how accurate the Kalman filters are performing. In some agent-based frameworks that may use this Kalman based prediction system, it may be acceptable if 50% of the predictions are within the percent error tolerance while in other frameworks it may need to be more accurate to be effective.

Dynamic Routing Queue Controller Motivation

The Dynamic Routing Queue Controller (DRQC) is a centralized network controller that reroutes network flows based on flow priority, queue predictions, and network congestion. When the controller detects congestion in the network, the controller adjusts the network to eliminate the network congestion. Network congestion is where the queue size is above normal. This section describes the design of the DRQC, which controls network flows to optimize a network according to the current and predicted states of the network.

With Kalman filter predictions, it is now possible to create an application that can utilize them. This application can be used to optimize a network and its flows so that there is reduced network congestion. This controller is a centralized system where it has

access to queue sizes, predictions, and network flows. A benefit of a centralized system is the ease of maintaining accurately updated lists of data that can be easily accessed from all points. A weakness in this type of system is that it is centered on a few components. If those central components fail, then the system as a whole is affected greatly. A distributed system can fix this problem but would make things more complicated to simulate. The overall goal of this controller, though, is to show that these Kalman filter predictions can be used to optimize a network, therefore a centralized system is good enough.

A network flow is a sequence of packets from a source node to a destination. In this controller, a network flow has minimum bandwidth requirements. When there is not enough bandwidth available on the flow's current path, it is to be rerouted if possible or paused so other network flows can utilize the network capacity. The network flows in this controller have a constant arrival rate, which means the packets are generated at a consistent average rate.

A key feature of this controller is that it is designed for prioritized network flows. *Prioritized network flows* are flows where they are ranked in order of importance. Higher ranked prioritized network flows should be given the available bandwidth in the network before lower ranked prioritized network flows. This allows network flows that are vital to the mission or overall network work goal to be given a high ranked flow. Through this setup, vital mission flows have a better chance to make it to its destination and fulfill quality of service requirements for that flow. If there is limited bandwidth in the

network, lower priority flows will be rerouted or stopped completely until there is more bandwidth available.

DRQC Features

Through the motivation of the DRQC, it is now possible specify the requirements that the DRQC needs to perform. These features spell out what the DRQC will need to do to optimize the network. These features will work with a scalable size network and network flows. In Chapter IV, there will be many simple simulations that will demonstrate these features in small networks to show that they are functioning. The five main features are listed in Table 3.2.

Table 3.2: Key Features of DRQC.

Priority rerouting
Priority flow control
Prediction detection
Dynamically split flows
Flow reactivation

Priority rerouting changes the network flow paths with respect to their priority. This is done by changing the routing tables in nodes. This requires a specialized type of

routing table because normal routing tables do not route according to a specific flow. Normal routing tables in routers only looks at the source and destination in the flow packet's headers. The routing tables in this controller need to look at source, destination, and flow ID so that the higher priority flows are processed first and let them get the optimal network paths first.

Priority flow control deals with starting and stopping flows according to priority. If there is not a path available for a network flow, then it will be stopped in order to allow higher priority flows to function. This allows higher priority network flows to have the available bandwidth before lower priority network flows. The DRQC will have the ability to adjust the network flows to adapt to the current network situation. This is a key requirement to prevent network congestion.

Prediction detection is the controller's ability to look at future queue size predictions from the Kalman filter. NS2 is programmed in C++, and the controller will have the queue data point from the link object. In reality though, it is assumed that controller will have direct access to the actual and predicted queue size data through a centralized system as explained earlier. On a non-Kalman filter link, the prediction is not available and the controller will only look at the actual queue size to see if there is congestion on the link. This feature reacts to the predictions through priority rerouting and priority flow control.

Dynamically splitting flows is the capability allows that the controller to split flows. The decision to add this feature to the controller was to allow more flows to be fitted through the network. The program that is going to be used to make the routing

decisions is called cost-scaling 2 (CS2) [20], which will be explained in the design section in this chapter. With CS2, most of the work involved routing is already implemented and the DRQC only needs to make the logical operations to change the routing tables and the ability to create separate network flows to match the CS2 decisions. This allows network flows to squeeze through a network within the limited available bandwidth. Part of this requirement is the controller's ability to keep track of newly formed flow ID's. The routing table mechanism looks at source, destination, and flow ID. To make multiple flows, you must make unique flow ID's.

Flow reactivation deals with starting flows that were stopped to prevent network congestion. Each flow has a start time and end time. If a current active network flow finishes sending data and there are network flows that were stopped in order for that network flow to work, then the DRQC will need to reevaluate the network condition to see if those stop network flows have enough bandwidth at the current to time to be reactivated. This lets lower priority flows to function after the higher priority flows have finished streaming.

DRQC Design

The basic design of the DRQC is that it receives network conditions and produces decisions to optimize the network which is illustrated in Figure 3.2. The network conditions are the actual and predicted queue sizes in the nodes in network. This process happens at every prediction time. For instance, if the Kalman filter is set to make a predictions about the queue size five seconds in the future, then this process will happen

every 5 seconds also. The DRQC will make decisions for the network based on the actual and predicted queue sizes. If there is little or no congestion detected, then the DRQC will make no changes unless there were network flows paused or rerouted due to previous network congestion. If there is congestion or network flows manipulated by the DRQC earlier, the DRQC will change routing tables and adjust network flows so that the congestion is fixed and updated. Either the DRQC will produce a set of instructions to execute so that the network will be improved or make no changes to the network because there is no network congestion. These instructions produced by the DRQC will be through TCL commands, which are executed into ns2.

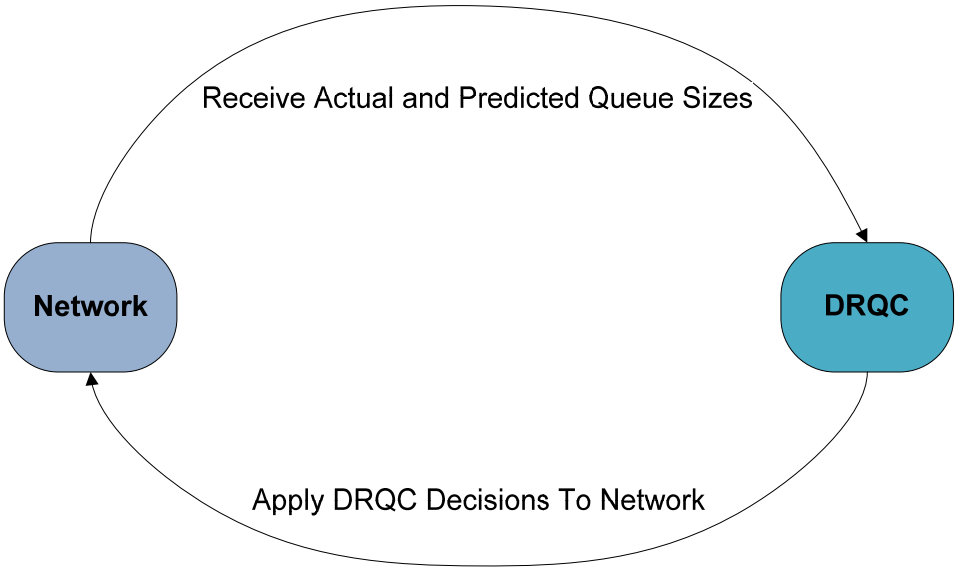

Figure 3.2: State Diagram of DRQC.

At each time prediction, the DRQC receives the actual and current queue sizes from each queue in the network. Before going through the process of assigned routes to the network flows, the DRQC goes through a series of logical decisions to determine how to handle the current network situation which is displayed in Figure 3.3. At every time a prediction is made, the DRQC will go through this process like previously mentioned in the state diagram in Figure 3.2.

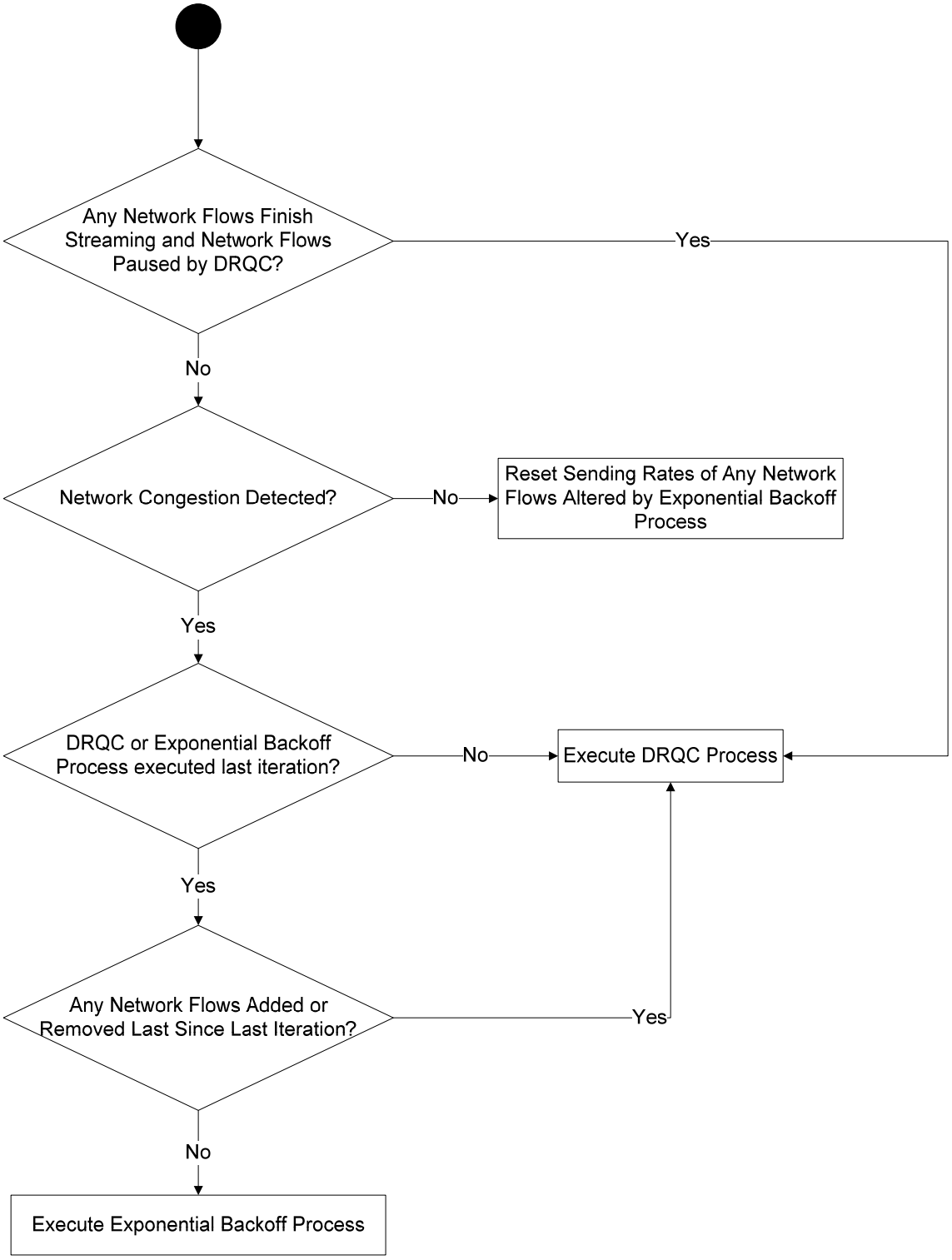

Figure 3.3: Flowchart of DRQC when it receives Kalman filter queue predictions.

The first logical decision is to determine if there were any network flows that finished streaming and if there were any network flows that were altered by the DRQC process. If there were any network flows altered, then the network flow was either paused or rerouted to a less optimal network path. This provided the optimal paths to the higher priority network flows. If this logical decision is true, then DRQC rerouting process will be executed to check to see if there are better network paths available due to network flows finishing from streaming.

The next logical decision is to determine if there is any congestion in the network. Experiments in Chapter IV were used to determine what exact conditions of the queue would indicate congestion in the network. It was concluded that if a queue had a queue size that was half full or if the future queue size is predicted be also half full, than the next time step would result in a full queue or long enough delay to cause problems in the network flow. Keep in mind that having any packets in a queue reduces the effectiveness of the link exponentially [21]. This is because a packet must wait in line for each packet before it, assuming that all the packets have the same service time. For example, suppose we have a link with 100 megabytes/second of capacity and has 5 packets on the queue. Each of these packets is the size of 1000 bytes. Every packet must wait the amount of time it takes for a packet to come across the link for each packet ahead of it in the queue. By using the equation in Equation 3.1, the 100 megabytes/second is now performing like a 20 megabytes/second by just having 5 packets on the queue. Therefore, with the queue being at least half full or the predicted queue size of it being half full, it causes enough

limited bandwidth in the link where the DRQC needs to intervene and fix the network congestion.

$$link\ performance = \frac{(packet\ size)}{\left(\frac{(packet\ size)}{(original\ capacity)}\right)(number\ of\ packets\ in\ queue)}$$

Equation 3.1: Link performance equation when there are packets in the queue [21].

If there is network congestion, then the DRQC goes on to the next logical decision. If there is no network congestion, then the DRQC makes no changes to the network routing tables or network flows. The DRQC also resets the sending rate of all network flows to its original rate. The sending rate of a network flow is adjusted when there is still congestion in the network and nothing was changed in the network conditions, which is explained later in this section.

If there is network congestion detected, the next thing to do for the DRQC is to figure out how to fix it. The first logical step in figuring how to fix the network congestion is to determine if the DRQC routing process was executed in the last time iteration. The DRQC routing process adjusts the network routing tables and flows so that there is no congestion in the network, which is explained in more detail later on in this section. If there is network congestion and the DRQC routing process was not executed, that means the network situation was changed and the DRQC routing process must be executed to fix the problem.

If there is network congestion and the DRQC was executed in the last time iteration, then the next logical decision is to see if there were any changes in the network situation such as added or removed network flows. These changes in the network must have caused the network congestion and must be altered to ensure the network flows of high priority are guaranteed to meet its QoS requirements. When changes are detected in the network situation and there is network congestion, then the DRQC will execute the routing process to fix that congestion.

In normal queuing, a small number of packets in a network queue are common; therefore in these types of networks, a certain accepted queue size tolerance can be set. For these experiments, though, the network traffic is constant and there should not be any packets in the network queue when there is no network congestion. If there was no changes in the network situation and the DRQC routing process was executed last time iteration, then that means that the solution that was issued by the DRQC did not fix the network congestion. Assuming there are no link or node failures, this is caused by network queues that still have packets in them. As explained earlier, when there are packets in the queue, this causes severe harm to the link performance. When the link's capacity is filled to the maximum capacity, it doesn't allow the queue size to return to zero. Instead, the queue stays at the same queue size due the number of packets entering the queue is equal to the number of packets exiting the queue.

The approach of solving the problem of a stagnant queue size due to links being at full capacity is to multiplicatively decrease the sending rate of the network flows. This method finds an acceptable sending rate for the queues to return to normal. This

approach is *exponential backoff* and is also used by carrier sense multiple access with collision avoidance (CSMA/CA) and carrier sense multiple access with collision detection (CSMA/CD) networks to retransmit frames. This is superior method than temporarily pausing the network flows for the amount of time to allow the queues to clear. A pause time for network flows could cause too much disruption in the application that uses the network flow. It would be better to decrease the sending rate temporarily so that the queue sizes can return to normal. The process of using exponential backoff is shown in the flowchart in Figure 3.4.

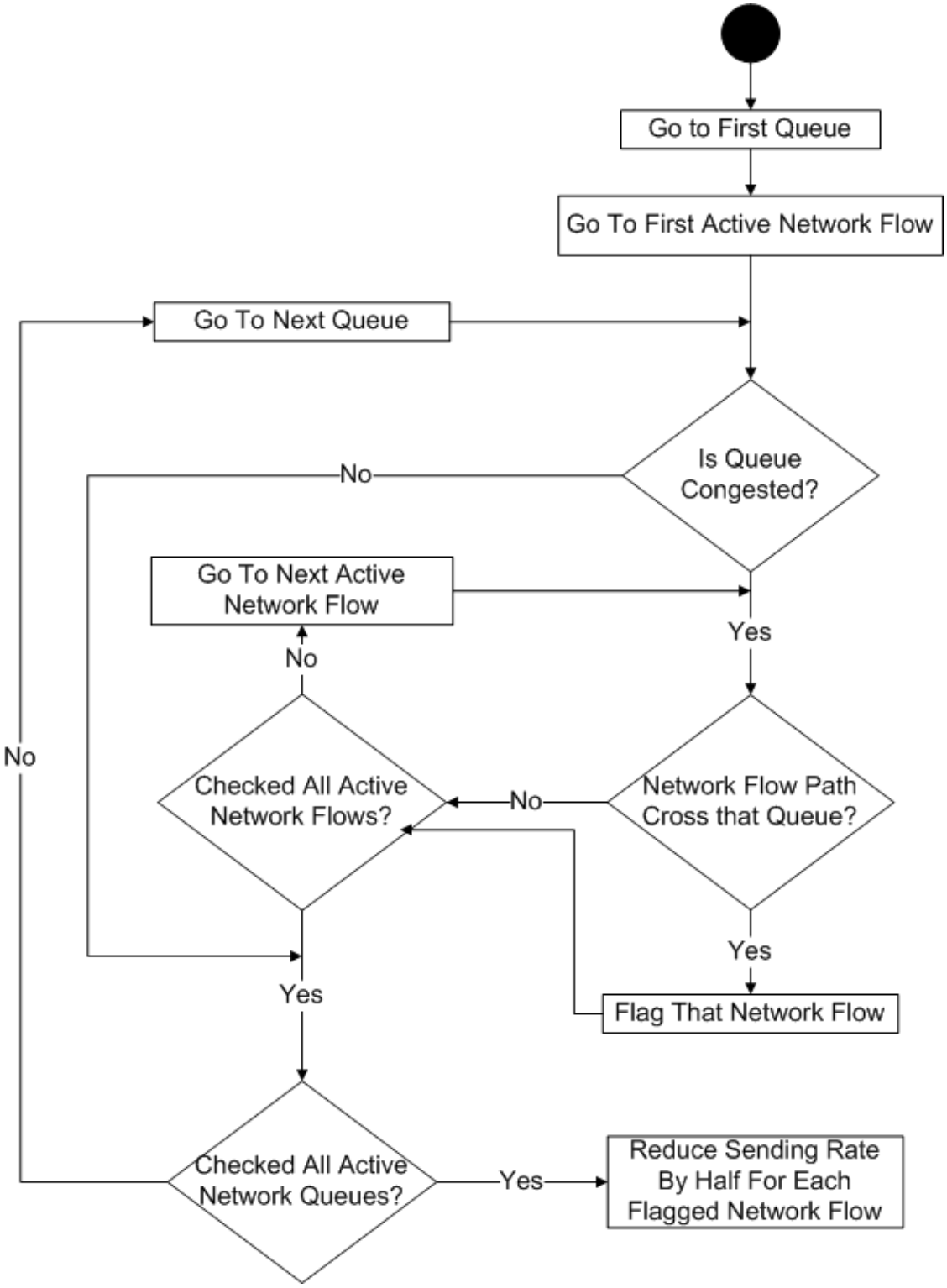

Figure 3.4: Flow chart of the exponential back off process.

When the exponential back off process is executed, the DRQC goes to the first queue and first network flow. The order in which the network flows and queues are processed is irrelevant because all the network flows that are causing this network congestion on a particular queue will have its sending rate decreased. A network flow can be causing congestion at multiple queues but a network flow can only be flagged once. This allows a network flow's sending rate to be reduced once during this process. After the DRQC goes to the first queue and first network flow, the DRQC starts to cycle through the queues and stops when a congested queue is found. When a congested queue is found, the exponential back off process goes though the active network flow's paths. Each network flow has a path list to show where the packets hop throughout the network to get to their destination. If that queue is in that network flow's path, then that network queue is flagged. After this process has been executed on all the queues, the sending rate on all the network flows that were flagged is adjusted. The adjustment reduces the current sending rate of the network flows by half. This allows the queue sizes to reduce to zero for the next time iteration if no other network flows are added before then.

The other outcome if network congestion is found in the network is the execution of DRQC's rerouting process. This process results in network flows being stopped, started, and rerouted to eliminate the network congestion. This process takes into consideration the network flow's priority, current path, and bandwidth requirements. With the predicted and actual queue sizes, the DRQC knows where the congestion is

located in the network. Using this information, the DRQC routes the network flows in order of priority. This process is shown in the flowchart in Figure 3.5.

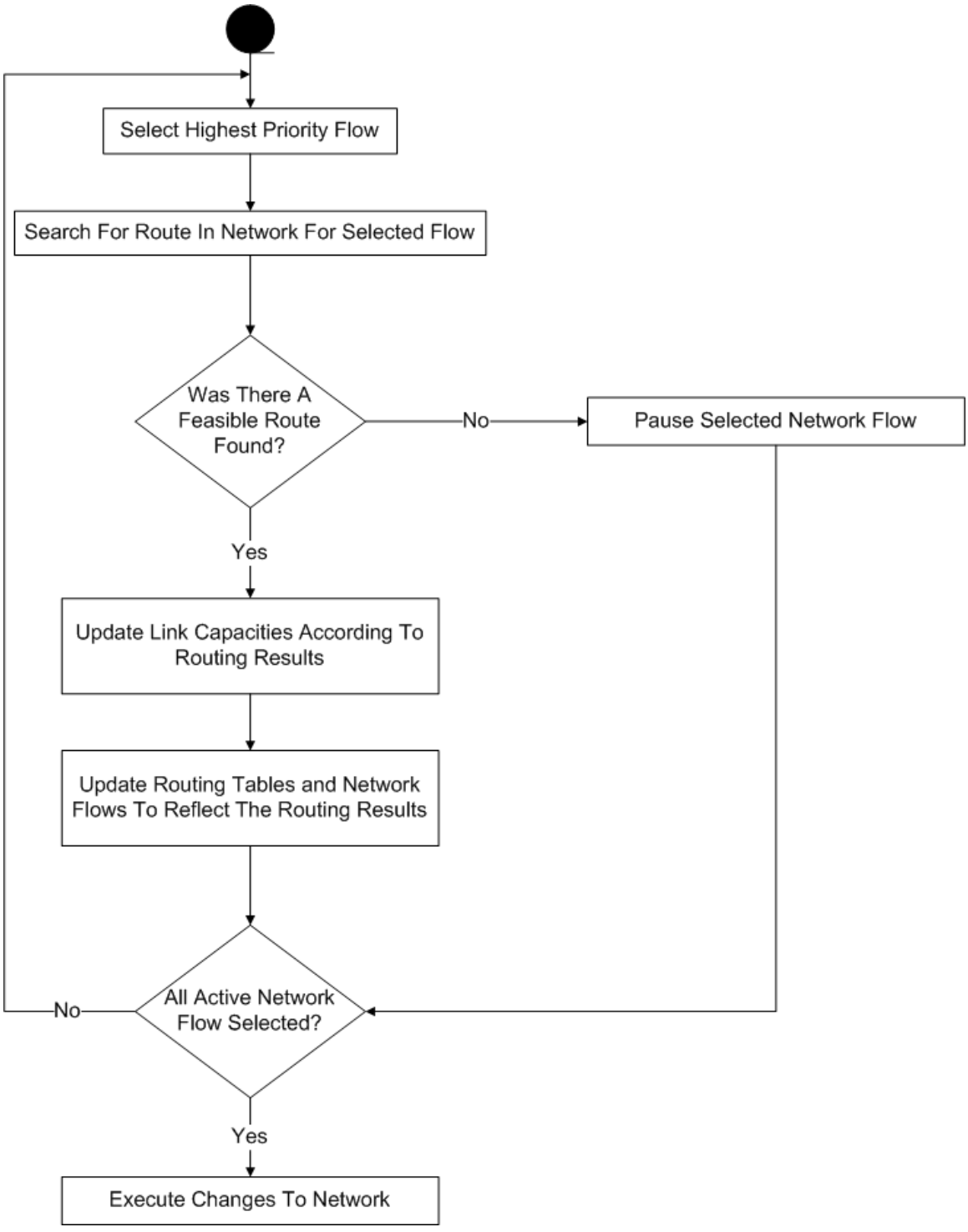

Figure 3.5: DRQC rerouting process flowchart.

The first operation that the DRQC rerouting process does is finding the highest priority flow. If two flows have the same priority, than the flow with the higher bandwidth requirement is selected. If both flows have the same priority and same bandwidth requirement, then the flow that has been active the longest will be selected. This flow is selected to be pushed through a network that contains the original link capacities.

The next operation is for the DRQC to make routing decisions for the selected network flow. DRQC uses an efficient implementation of a scaling minimum-cost flow algorithm through a program called CS2 [20]. CS2 finds the cheapest way for a flow to go from its source to destination. The input for CS2 is a network of links with their capacities and a network flow. The output of CS2 is a resulting network of updated links. When a flow goes through a link, the link capacity is reduced by the amount of bandwidth that the flow contains. For instance, if a flow requires 10 megabytes/second of bandwidth and the link has 15 megabytes/second of capacity, the updated link capacity will now have 5 megabytes/second of capacity.

If there is enough capacity in the links between the source and the destination, then the network flow will be pushed through network and the output. The output will show the resulting network link's capacities. The DRQC will take this information and find the path of each the network flow section. The DRQC also finds how many ways the flow is divided. The network flow will split if there is not enough capacity throughout the most direct path from source to destination. The DRQC updates the link

capacities according to the CS2 output. The next step is to change the routing tables in the nodes to reflect the network flow paths. If a flow was divided, then a separate flow needs to be created to reflect the decisions made from CS2. This is also accomplished by executing TCL commands in the NS2 engine.

If CS2 cannot find a feasible path for the selected flow, the output will be blank and CS2 will indicate that the network flow isn't able to be pushed through the network without causing congestion. The DRQC will pause that network flow to fix the network congestion. If there are network flows that were stopped, then a flag is activated so that if a high priority flow is used later. This process is run again to see if this flow can now be reactivated. This fulfills the requirement of flow reactivation and allows lower priority network flows run after higher priority network flows have finished transferring packets from a source to a destination.

This whole process is repeated until all the active network flows have been selected. This greedy process gives the highest priority flow the best chance to be pushed through the network without it being affected by lower priority network flows. Changing the routes of the network flows is made possible by changing the routing tables. The routing tables have next hop information for network flows that come from a certain source and destination node. In additional to this, the DRQC can make exceptions due to flow ID. This makes it so a network flow can be separated into two paths despite the network flowing coming from the same source and destination.

Summary

In conclusion, the main concept of the DRQC is that it monitors the current and predicted queue sizes until there is a congestion problem in the network. When there is a congestion problem that is predicted or is actually happening at the current time, then the DRQC will invoke a series of procedures to correct that problem or prevent it from even happening. This series of process's goal is to fulfill the requirements of the DRQC. The DRQC takes control of the routing tables in the nodes and the network flows to optimize the network and clear the network congestion. The DRQC makes intelligent decisions regarding how to handle network flows that have priority. The decisions the DRQC are based on the information it has available from the Kalman filter. These decisions are greedy and are based on the current network state.

IV. Analysis and Results

This chapter describes the experiments performed with Kalman filter predictions and the implementation of the DRQC that was mentioned in the previous chapter. The first set of simulations demonstrates how the Kalman filters perform in a more realistic network environment. While previous Kalman filter based network prediction research only used two nodes with multiple types, this chapter will look at using a more complex network with more network flows. The next section of the chapter will describes the implementation of the DRQC. There is a small set of simulations which demonstrate the key features of the DRQC. The DRQC is then implemented in a more realistic network which is used to demonstrate the performance of the DRQC.

Kalman Filter Validation Simulation

With NS2, a network of nodes has been generated that reflects the topology of a more realistic network as seen in Figure 4.1. The traffic generator randomly creates 85 Transmission Control Protocol (TCP) and 15 User Datagram Protocol (UDP) network flows. Each network flow has random sources and destinations assigned but are not unique as two network flows may have the same source and destination. The arrival rate of these network flows come from an exponential distribution, which means it is a process in which events occur continuously and independently at a constant average rate. That constant average rate was randomly selected in-between 0.16 and 0.84 megabytes/second for each network flow. Each link in Figure 4.1 is

duplex and has a bandwidth of 4 megabytes/second with a propagation delay of 25 milliseconds. Propagation delay is the amount of time it takes for the packet to travel from the sender to the receiver over a medium which is caused by physical constraints of the link.

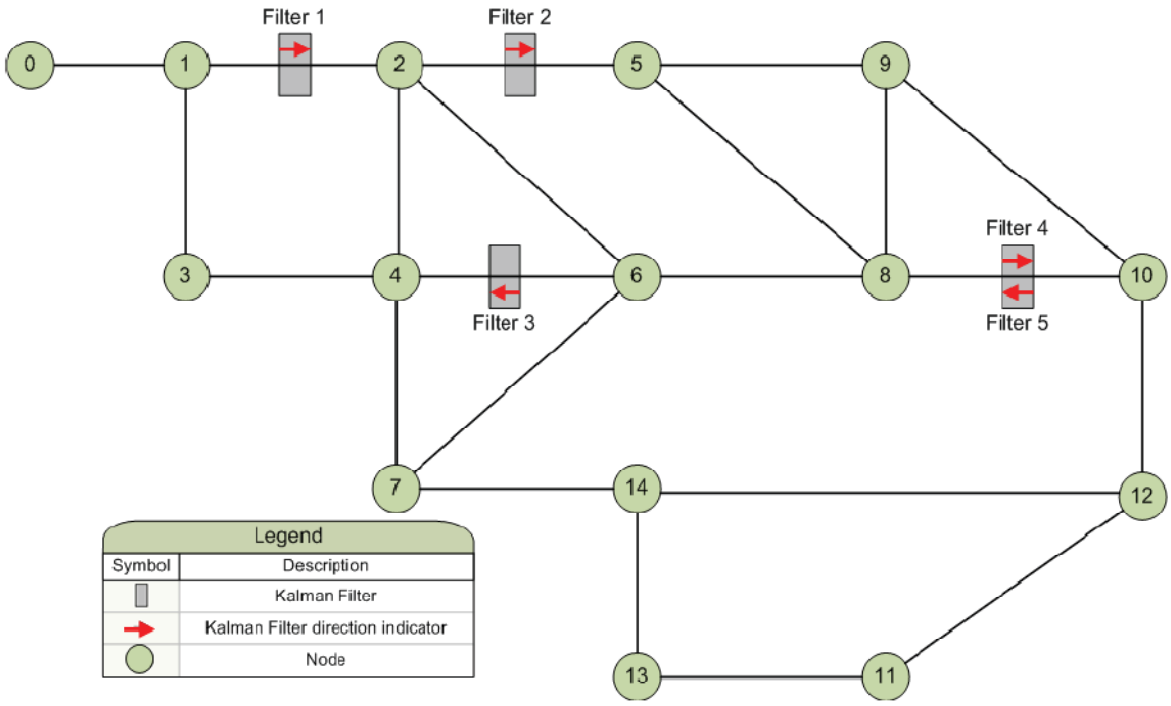

Figure 4.1: Kalman filter validation simulation network topology for the first set.

In order to expand the previous research mentioned in Chapter II, five Kalman filters are placed at strategic locations in an arbitrary network topology and where the network traffic is most likely to be routed to. These Kalman filter locations are shown in Figure 4.1. Keep in mind that the drop tail queue is outbound, and Kalman filter 4 belongs to node 8 and Kalman filter 5 belongs to node 10. All the queues are located outbound of the nodes and are drop tail, which is common in Internet routers. With

drop tail queues, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic. The maximum capacity of queues in this experiment is 1000 packets with each packet containing 1000 bytes. Each Kalman filter is set to predict 5 seconds into the future and will make a prediction every 1 second. This simulation is set to execute for 500 seconds in simulation time. Therefore, each Kalman filter will collect 500 predictions over the course of the simulation. These predictions will be compared with the actual queue sizes. The results from this experiment are taken into consideration for the design of the DRQC.

Kalman Filter Validation Simulation Results

This simulation was separated into two different sets. The first set is simulations that utilized more stable network traffic so that the queue sizes did not dramatically change. The first set's results was the averaged over 15 runs. These results used the previous research's methods of evaluating how well the Kalman filters are performing. It was done this way so that it was possible to compare the results to earlier simulations. These earlier simulations only used two nodes and multiple types of network flows.

Table 4.1: Results from the first set of Kalman filter validation simulations.

Kalman Filter	1	2	3	4	5
Actual Queue Size Average	32.846	6.962	3.665	7.922	64.358
Predicted Queue Size Average	33.826	7.703	4.633	8.256	60.782
Difference of Averages	0.980	0.741	0.968	0.334	3.576
Percent Error	2.984%	10.643%	26.412%	4.216%	5.556%

Table 4.1 contains the results from the first set of Kalman filter validation simulations. Originally, there were six Kalman filters with this simulation. One of the Kalman filters would error while executing and stop the simulation. The cause of this was that the predictions would be above the maximum queue size and cause an error in mid simulation. The solution to this problem was when a prediction was above the maximum queue size is to set the prediction equal to maximum queue. For instance, if the prediction was 1500 packets and the maximum queue size was 1000. This fix was used in the second set of Kalman filter validation simulations.

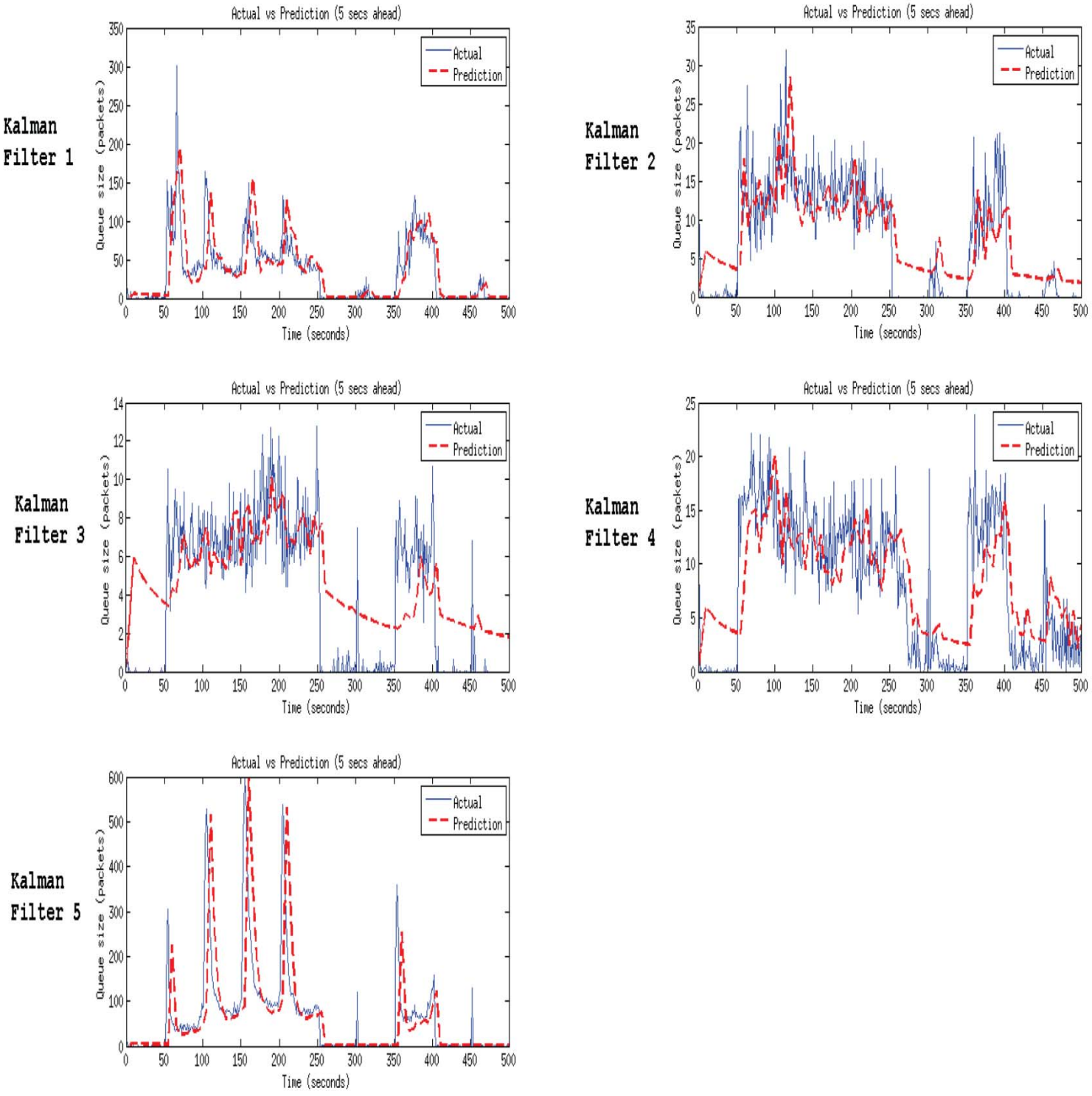

Figure 4.2: Graphs of the queue sizes during the simulation in first set of Kalman filter validation simulations.

In Figure 4.2, the graphs show the queue sizes in the Kalman filters placed inside the network. The solid blue line is the actual queue size and the red dotted line is the predicted queue size throughout the simulation. As you see from the two node case in Figure 2.6 from Chapter II, the results are similar from the previous research. There is a problem with these results, though, and that's the prediction red dotted line is about a whole prediction phase off from the blue line. This led to the inspiration to a second set of Kalman filter prediction simulations.

One of the hypotheses on why the prediction data is a time phase off was that 5 seconds is too long of a time to make accurate predictions. This seems reasonable because there has to be some evidence to indicate that the queue size is going to change in order for the predictions to be correct. So for this Kalman filter validation simulation, the Kalman filters are set to make a 1 second prediction every 1 second. The Kalman filter with the queue that was always near full during the simulation was also fixed and is added between node 7 and 14 as shown in Figure 4.3. The network traffic was also toned down so that queue size would become burstier to make the network more realistic. This causes the queue size to change more dramatically.

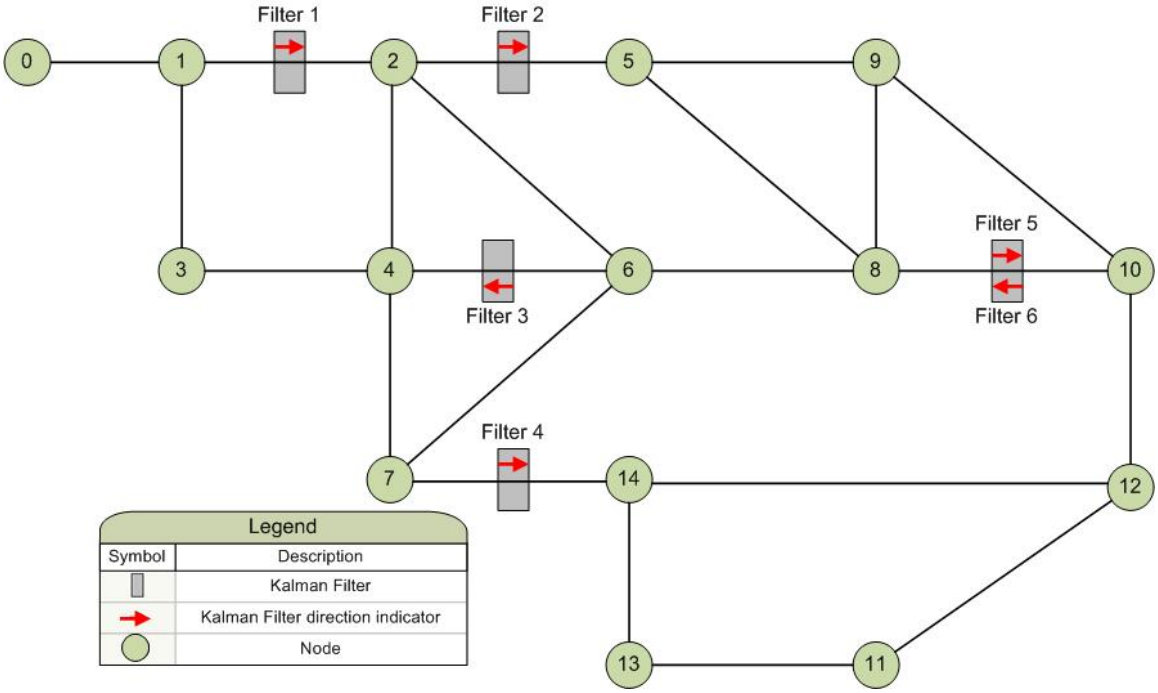

Figure 4.3: Kalman filter validation simulation network topology for the second set which has a sixth Kalman filter.

Like in Figure 4.2, Figure 4.4 shows the queue sizes in the Kalman filters placed inside the network. The solid blue line is the actual queue size and the red dotted line is the predicted queue size throughout the simulation. These results are similar to the first set in that the prediction queue size line seems to be one time phase off from the actual queue size line.

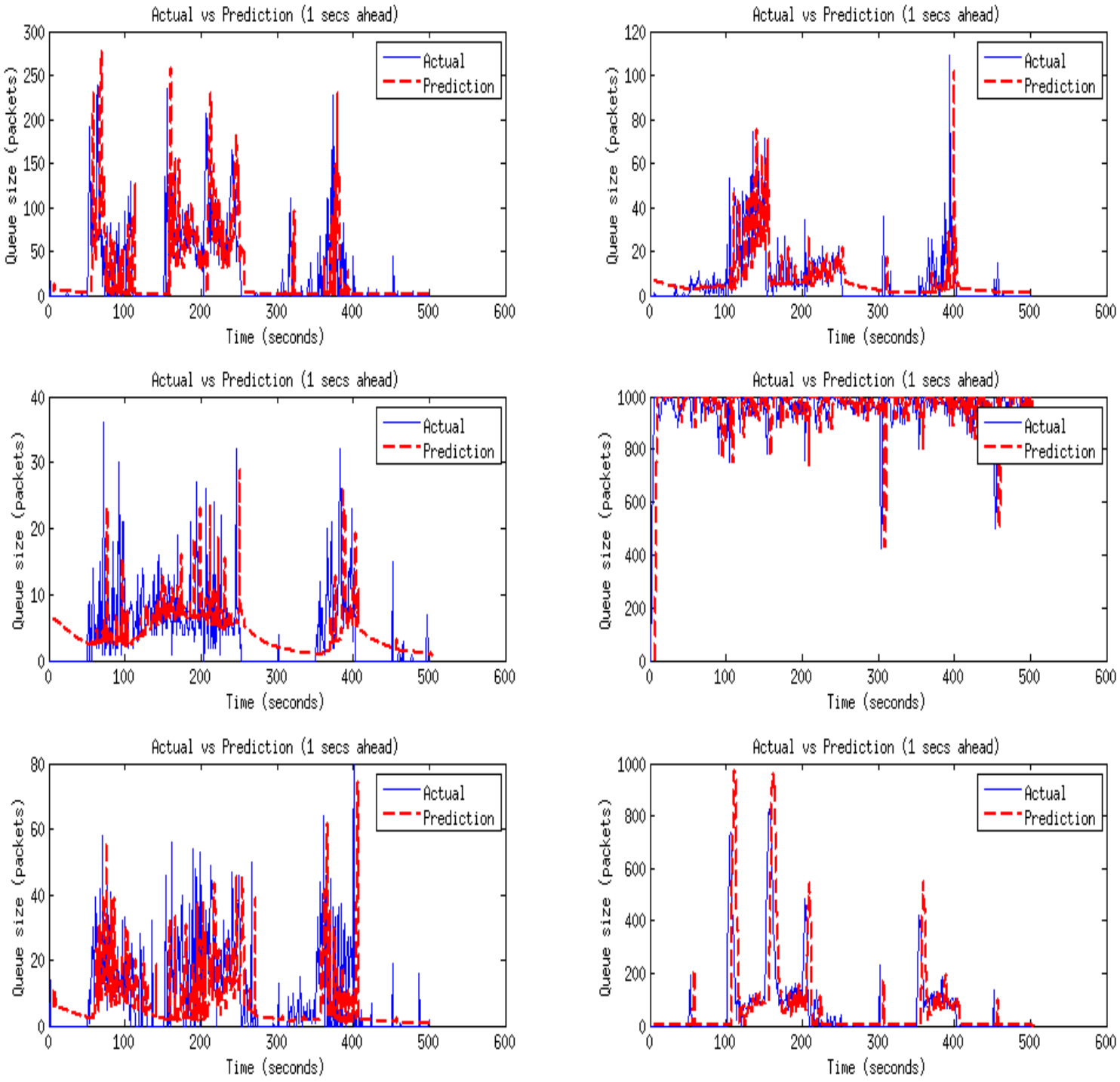

Figure 4.4: Graphs of the queue sizes during the simulation in the second set of Kalman filter validation simulations.

In Figure 4.5, it shows the actual queue size versus the predicted queue size of Kalman filter 1 from the second set of Kalman filter validation simulations. The blue line is the absolute value of the difference between the actual queue size and the predicted. The closer to zero that blue line is, the more accurate the predictions are. As you can see throughout the simulation, the difference through the simulation is about 10 packets with spikes up to 30 packets. Considering the maximum queue size is 1000 packets, 30 packets is not that significant.

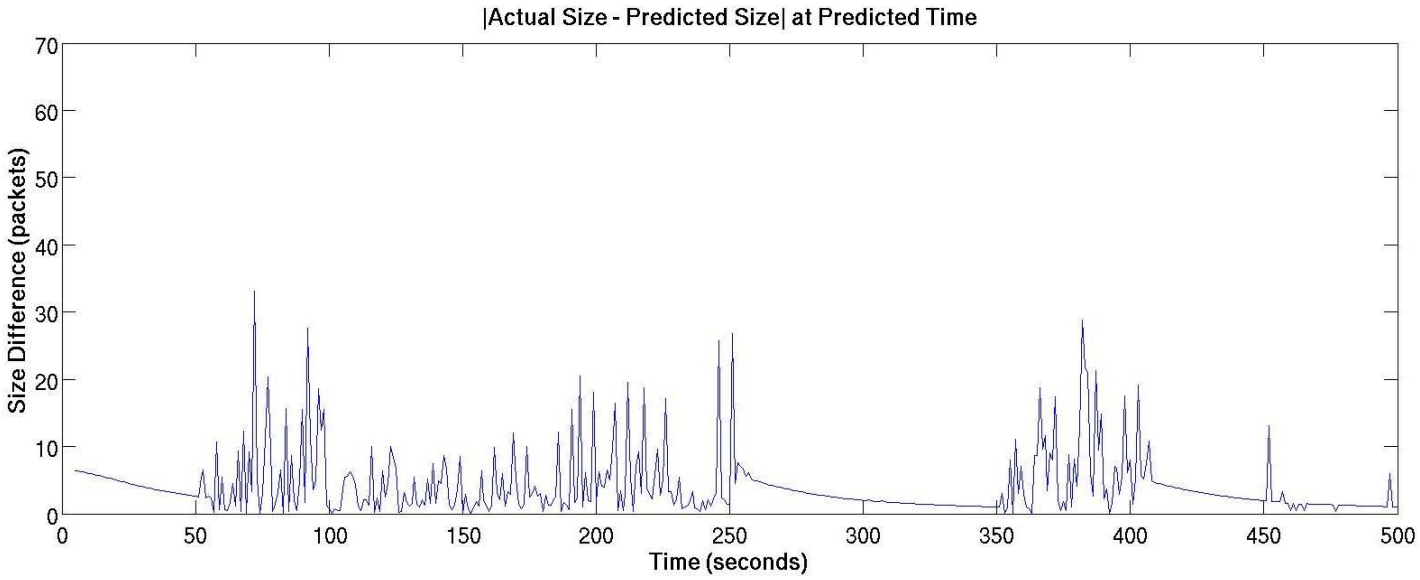

Figure 4.5: Actual Queue Size vs. Predicted Queue Size Graph of Kalman filter 1 in the second set of Kalman filter validation simulations.

In Table 4.2, the results from the second set of Kalman filter validation simulation is shown. These results use the Mean Average Percent Error (MAPE) and Kalman Rating methods to evaluate the performance of the Kalman filters. These methods were explained in Chapter III. The performance with these methods

concludes that Kalman filter predictions are very poor when the network traffic is very bursty like in this simulation.

Table 4.2: Results from the second set of Kalman filter validation simulations.

Kalman Filter	1	2	3	4	5	6
Actual Average Packets	25	7.5	4.6	963.9	11.55	50.83
Predicted Average Packets	25.25	7.6	5.1	963.8	10.59	50.31
Kalman Rating (80%)	70%	66%	57%	11%	84%	88%
MAPE	282%	81%	44%	5%	153%	838%

An original idea with the previous research on these Kalman filter predictions was to use them to reassign link capacities according to the predicted queue sizes. With these newly reassigned link capacities, a network controller could be used to fix any problems in the network before the problem actually occurred. In order for this to be possible though, the Kalman filters must be very accurate. These simulations demonstrate that when using Kalman filters in a more realistic network, Kalman filters do not produce as accurate results when compared with previous research. Keep in mind that previous research only used two node simulations to demonstrate the Kalman filter performance.

Although the queue size predictions are not accurate enough to reassign link capacities like previous research's vision, it may be possible to use these predictions to at least notify the network controller there will be a congestion problem in the future. This may be especially possible in a more controlled network environment where steady network flows are used and can be controlled. This is what the next section

will answer and will see if it is possible to use these predicted queue size predictions to notify a network controller that there will be network congestion in the near future. This will allow the network controller to fix the problem before it occurs.

DRQC Simulations

This section explains simulations that were used to demonstrate the functionality of the DRQC. The network environment in this simulation is assumed to be controlled with steady network flows. The DRQC has the ability to change routing tables in nodes and the network flows. In order for the DRQC to modify the routing tables in ns2, TCL functions are implemented into the simulation script. The routing tables have the ability to make specific routes for certain flows and route according to flow ID.

In order to demonstrate the functions of the controller, a series of small simulations made to demonstrate the features of the DRQC. With the success of the smaller simulations, they will demonstrate and fulfill the requirements of the DRQC mentioned in Chapter III. Figure 4.6 shows the network topology design for the first DRQC simulation. The circle nodes represent a source or destination for a network flow. The node name prefix *s* represents source and *d* represents destination. The square nodes are routers nodes and have the prefix *r* in their node name. The router nodes have the ability of changing routing tables to alter network flow paths. The ability to demonstrate the priority rerouting and priority flow control requirements will be demonstrated through this basic network setup. The different types of flows

through the network can demonstrate the flow control needed through the first set of simulations.

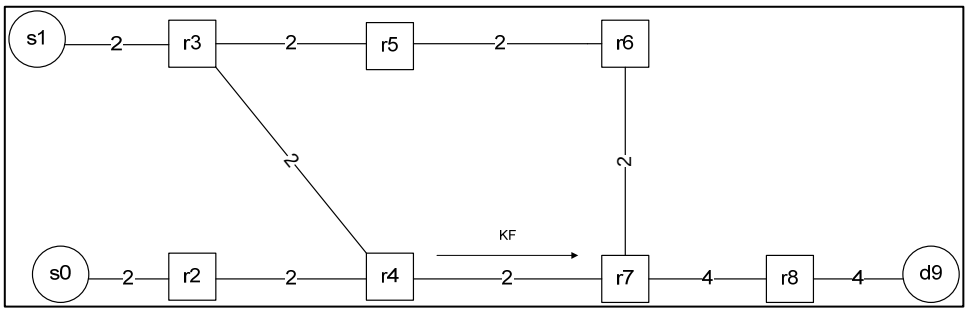

Figure 4.6: Network topology for the DRQC priority rerouting simulation.

The purpose of this simulation setup is to demonstrate rerouting of packets based on priority. The network contains 8 routers, 10 duplex links (two simplex links facing opposite direction between nodes), 2 sources nodes, and 1 destination node. The router nodes have the ability to change their routing tables dynamically. The source and destination nodes can be thought more as individual applications inside of the router node that it is attached to. There can only be one link attached to source and destination node. This makes the simulation more realistic because you cannot attach the same application process to two computer nodes. There can, however, be multiple source/destination nodes attached to a router node. There also can be multiple flows coming in and from source and destination nodes. The arrow between node r4 and r7 indicates that link is a Kalman filter drop tail link. Based on the conclusions from the Kalman filter validation simulation, the Kalman filter will be set to predict the queue size 1 second in the future.

DRQC Simulation: Priority rerouting

The purpose of this simulation is to demonstrate the DRQC feature of priority rerouting. There are two flows that are made for this simulation. This simulation will make the lesser priority flow to take the longer path to its destination due to network congestion at the Kalman filter. The flows are shows in Table 4.3 below.

Table 4.3: The network flow table for the DRQC priority rerouting simulation.

Flow	Priority	From	To	Bandwidth	Start Time	End Time
0	1	s0	d9	2	2	10
1	2	s1	d9	2	0	10

At the beginning of the simulation, network flow 1 will route by the shortest number of hops to node d9. Therefore, the initial route for flow 1 is going to be r3->r4->r7->r8->d9. When flow 0 starts at 2 seconds into the simulation, it will take the route of r2->r4->r7->r8->d9. This will result with a congestion of packets between r4 and r7. The key result in this simulation will be that flow 1 will reroute its packets to r3->r5->r6->r7->r8->d9 because of its lower priority to flow 0. Another result from this simulation will be the reaction time to the prediction of the future queue size. The ideal result will be that the packet flow from flow 1 will start rerouting before the queue size is filled and starts dropping packets.

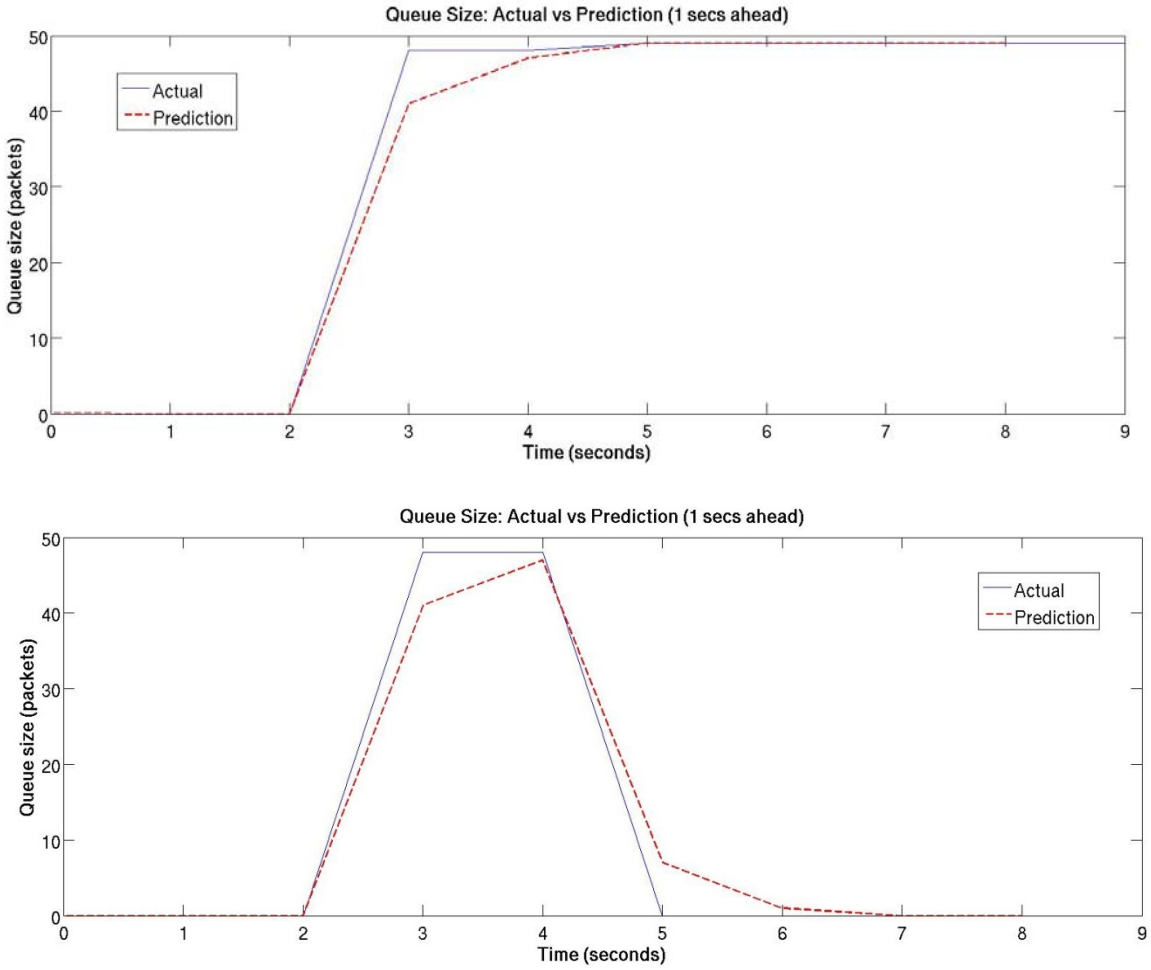

Figure 4.7: Graph of queue size in node 4 without using (Top) and with using (Bottom) DRQC.

The results from this simulation were successful in rerouting the network flow according the DRQC decisions. At 3 seconds into the simulation, the DRQC recognized the network congestion at node 4 by seeing the queue size was near full. The DRQC reroutes flow 1 to the longer path while not disrupting flow 0. At 4 seconds into the simulation, the DRQC recognizes that there is still network

congestion at node 4. This is because the network flow 0 takes the full capacity of the links and does not allow the queue to empty. The DRQC uses the exponential back process to decrease the sending rate of network flow 0 to 1 megabytes/second (half sending rate) while the other network flow is still at 2 megabytes/second. At 5 seconds into the simulation, the DRQC saw that there was no congestion and no changes were made with the network flows, therefore the DRQC restored the sending rate of flow 0. This was observed using the network animator (NAM) which is the visualization tool in ns2. The NAM visualization is shown in Figure 4.8. A NAM file is produced at the end of the simulation. The actual and predicted queue size data was also produced in a text file, for analysis.

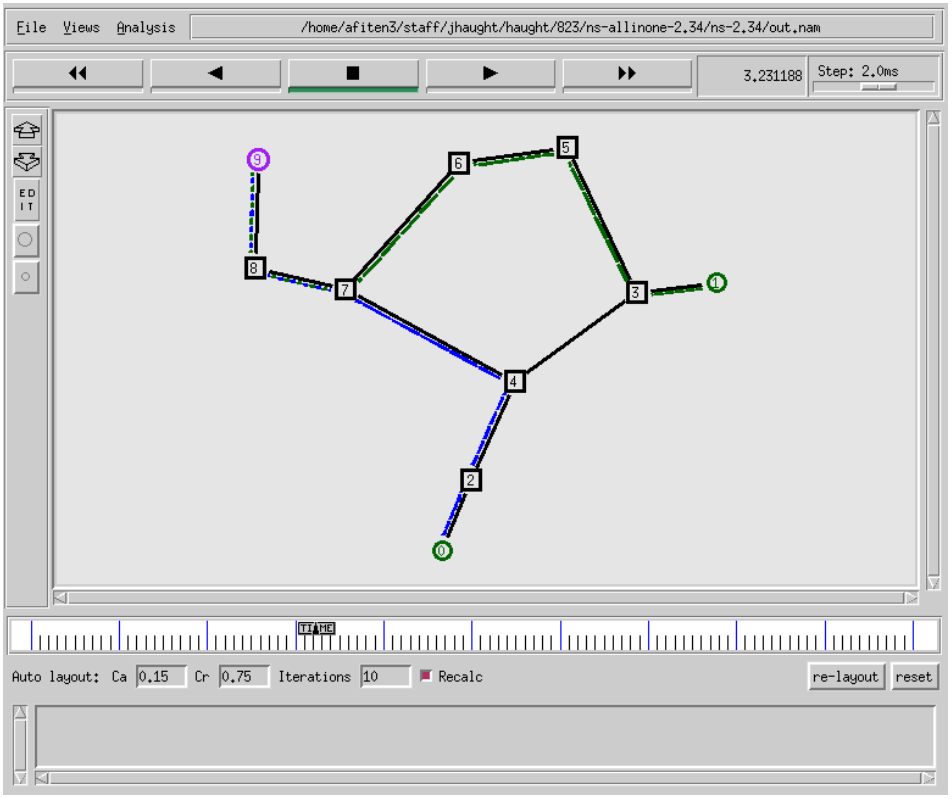

Figure 4.8: Visualization of the priority rerouting simulation using NAM.

DRQC Simulation: Priority Flow Control

In this simulation, the goal is to demonstrate the priority flow control requirement of the DRQC. The priority flow control mechanism of the controller stops flows of lower priority and not enough capacity in the network to provide a path for the flow. This shows that the DRQC has the ability to recognize network congestion and will stop a network flow if a feasible flow cannot be found. This simulation uses the same network topology as the previous simulation except that the link between s1 and r3 has a link capacity of 4 megabytes/second instead of 2 megabytes/second.

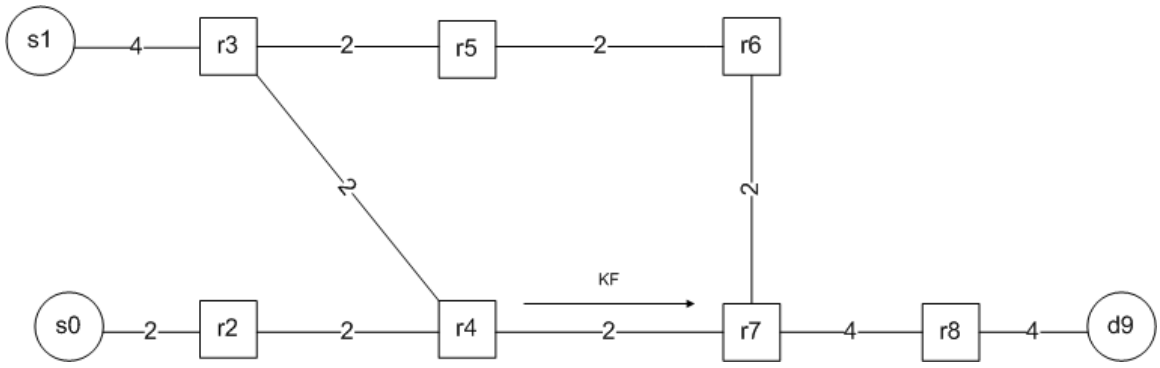

Figure 4.9: Network topology of priority flow control simulation.

This network topology was changed so that a flow can fill both available paths to d9 and prevent another network flow to be used without causing congestion in the network. These network flows makes it so that the flow from s1 splits into 2 paths. The flows is listed in the flow table is in Table 4.4.

Table 4.4: The network flow table for the DRQC priority flow control simulation.

Flow	Priority	From	To	Bandwidth	Start Time	End Time
0	1	s0	d9	2	5	10
1	2	s1	d9	4	0	10

At the beginning of the simulation, there is a network flow started at s1. This network flow will take the path s1->r3->r4->r7->r8->d9 because path with fewest number of hops between source and destination. There will be congestion at r3 because the link can only handle 2 megabytes/second. The DRQC will see this and make two network flows for that flow and each flow will have a bandwidth of 2 megabytes/second and fulfill the 4 megabytes/second requirements for flow 1. Later in the simulation, flow 0 will activate and cause congestion at r4. The DRQC will stop network flow 1 because it has lower priority than network flow 0. Network flow 1 will not have a path available and will be forced to stop. This will demonstrate the priority flow control requirement of the DRQC.

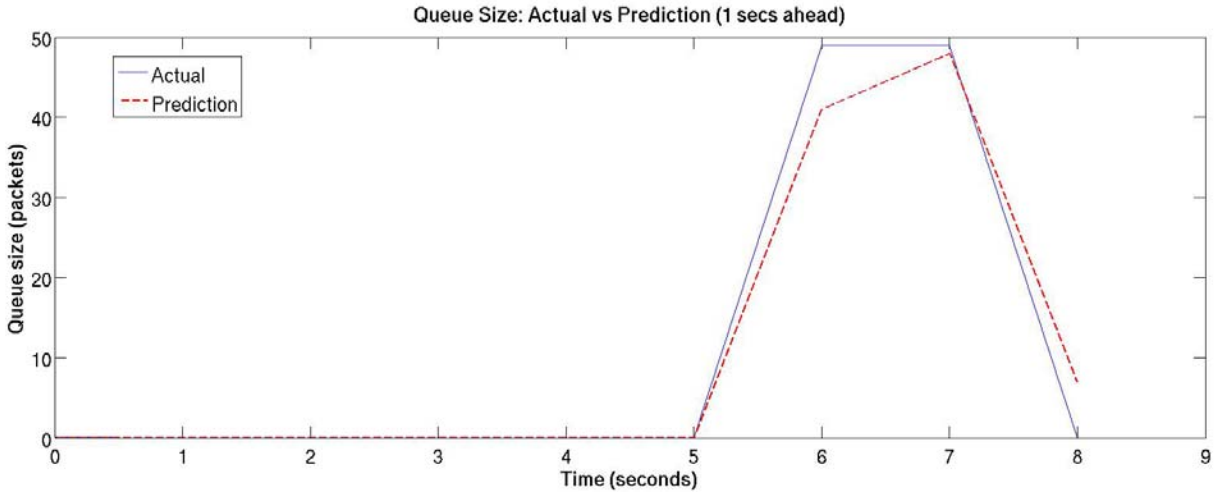

Figure 4.10: Graph of queue size in node 4 during the priority flow control simulation.

The results of this simulation was that it was successful in controlling the network flows by stopping network flow 1 and allowing network flow 0 to stream without interruption from network congestion. This was allowed because network flow 0 had a higher priority than network flow 1. At the beginning of the simulation, network flow 0 starts from s1 and packets are being immediately dropped at r3 because the queue size reaches its maximum which is shown in Figure 4.11. At 1 second, the DRQC makes 2 network flows for flow 0 to prevent the network congestion at node r3. The two paths is s1->r3->r4->r7->r8->d9 and s1->r3->r5->r6->r7->r8->d9. At 2 seconds into the simulation, the DRQC recognizes that there is congestion at r3 because packets haven't been given a chance to empty. Exponential backoff process is activated at 2 seconds into the simulation. The result of this is that the network flow's sending rate contains the path s1->r3->r4->r7->r8->d9 is reduced by half while the other alternative flow resumes its original sending rate. At 3 seconds, the sending rate is returned to normal because the network flows hasn't

changed. At 5 seconds, the other network flow is activated and packets are dropped at node r4. At 6 seconds, the DRQC recognizes this network congestion and deactivates the network flow 1 because it has lower priority and was unable to find a path in the network for it. At 7 seconds, the DRQC uses exponential back off to let the queue size empty at node r4. The rest of the time, network flow 0 streams without interruption from network congestion while network flow 1 is not active for the rest of the time.

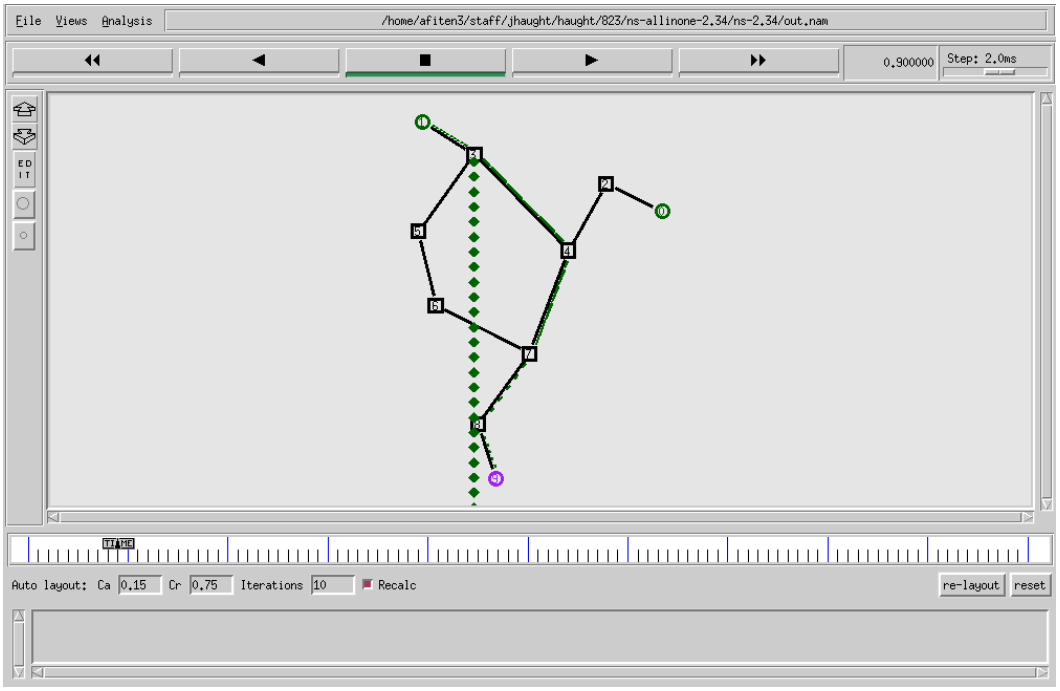

Figure 4.11: Visualization of the priority flow control simulation using NAM.

DRQC Simulation: Dynamically Splitting Flows

The purpose of this simulation is to demonstrate the ability to reroute a lower priority network flow and to split it within same time step. This allows the higher priority to take the shortest path to its destination. Lower priority network flows are forced to take the non-optimal paths or is temporary paused until the high priority network flows have finished streaming. If a lower priority network flow is forced to take the lesser optimal path but takes too much capacity for that path, then the DRQC has the ability to split that network flow into another path if there is one available. This allows that lower priority network path to fulfill its QoS requirements. The network topology is shown in Figure 4.12.

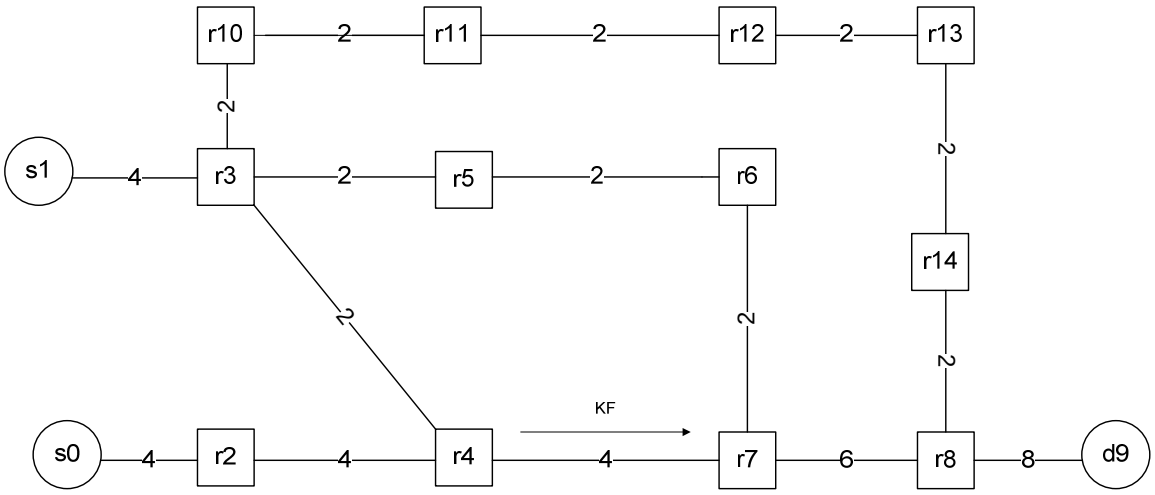

Figure 4.12: Network topology of DRQC dynamically splitting flows simulation.

The network topology for this simulation is similar to the previous ones except that there is an extra path provided to s1 to d9 and there's an increased link capacities at r8->d9 and r7->r8. This allows an additional alternative path for a network flow coming from s1 to d9. This alternative path is has a greater number of hops and wouldn't be the first choice. The list of network flows is listed in Table 4.5.

Table 4.5: The network flow table for the DRQC dynamically splitting flows simulation.

Flow	Priority	From	To	Bandwidth	Start Time	End Time
0	1	s0	d9	4	3	10
1	2	s1	d9	4	0	10

With these network flows, the simulation will be like the previous simulation but instead of the lower priority flow completely stopping, the lower priority flow will be rerouting and separated into two different streams. The idea in this simulation is that flow 1 will take the shortest path r2-r4-r7-r8-d9 and congestion will be detected at r3. The DRQC will detect this network congestion and will make two separate flows with of flow 1 is going r3-r4-r7-r8-d9 while other half will go through r3-r4-r7-r8-d9. Later in the simulation, network flow 0 will start. Network flow 0 will take the path r2-r4-r7-r8-d9 and cause network congestion at node r4. Since network flow 0 has high priority this will force flow 1 to reroute due to its lower priority. Network flow 0 will take the alternative routes available and send half its flow to s1-r3-r5-r6-r7-r8-d9 and the other half to r10-r11-r12-r13-r14. Both available network flows will be able to

fulfill their QoS requirements while giving the higher priority flow the most efficient route available.

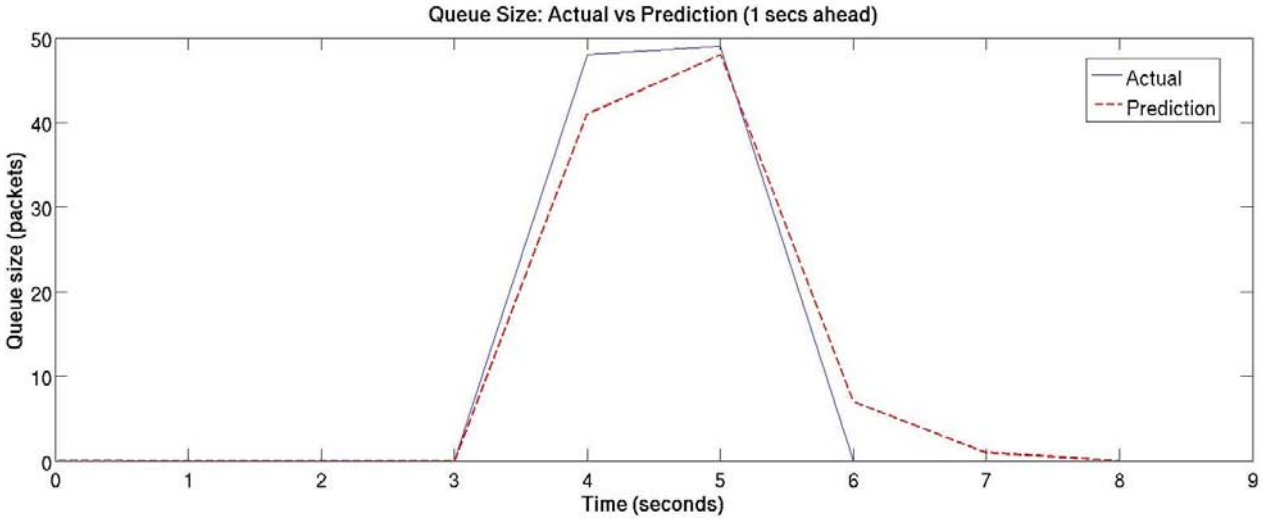

Figure 4.13: Graph of queue size in node 4 during the DRQC dynamically splitting flows simulation.

At the beginning of the simulation, network flow 1 starts and takes the route s1->r3->r4->r7->r8->d9. Packets start to drop at node r3, which is shown in Figure 4.14. At 1 second into the simulation, the DRQC detects the network congestion at node r4, and splits the network flows like explained above. At 2 seconds in the simulation, the DRQC sees that there are still packets stuck in the queue at node r3 and decreases one of network flow 1 stream's sending rates by half. At 3 seconds into the simulation, there is no more congestion at r3, the original sending rate is restored to normal, and network flow 0 has started and has taken the path s0->r2->r4->r7->r8->d9. This causes network congestion at node r4 and packets are dropped. This network congestion is shown in Figure 4.13 by the increased queue size at node r4. At 5 seconds into the simulation, the network flow 1 stream that took the path s1->r3->r4-

>r7->r8->d9 is redirected to take the path s1->r3->r10->r11->r12->13->14->r8->d9. This allows network flow 0, which is the higher priority flow than network flow 1, to take the optimal path while allowing network flow 1 to stream at a cost of taking the longer path. This simulation was successful in demonstrating the DRQC's ability to dynamically split network flows.

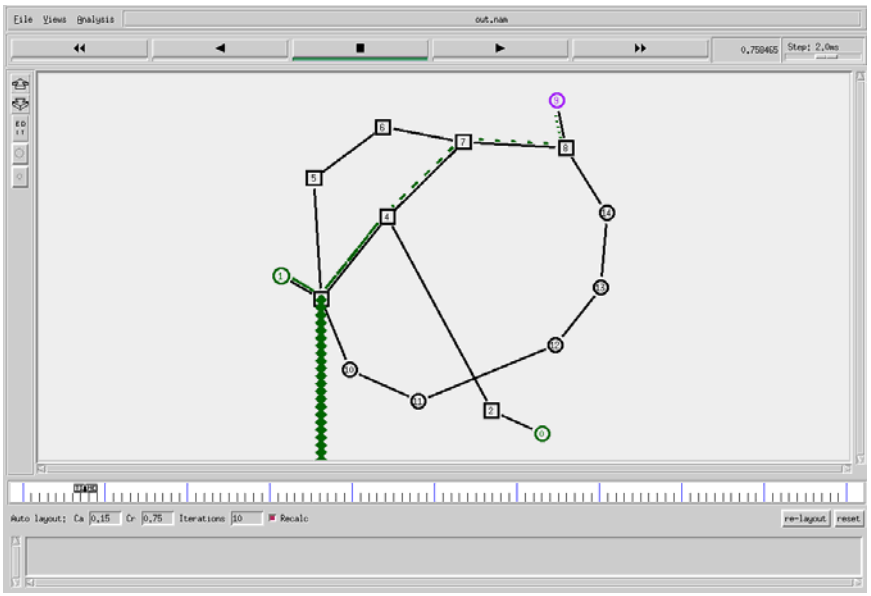

Figure 4.14: Visualization of the beginning of the DRQC dynamically splitting flows simulation.

DRQC Simulation: Flow Reactivation

The purpose of this simulation is to demonstrate the DRQC's ability to reactivate network flows that were paused due to network congestion. Once higher priority network flows have are finished streaming, there is extra network capacity available from where the higher network flows once occupied. At the time that there was higher priority network flows have stopped streaming and there were network

flows that were stopped because they had lower priority, there may be enough network capacity available for these network flows to be reactivated. So when this happens, the DRQC will run its rerouting process to check to see if there enough network capacity for those stopped network flows to be reactivated. The network topology for this simulation is shown in Figure 4.15.

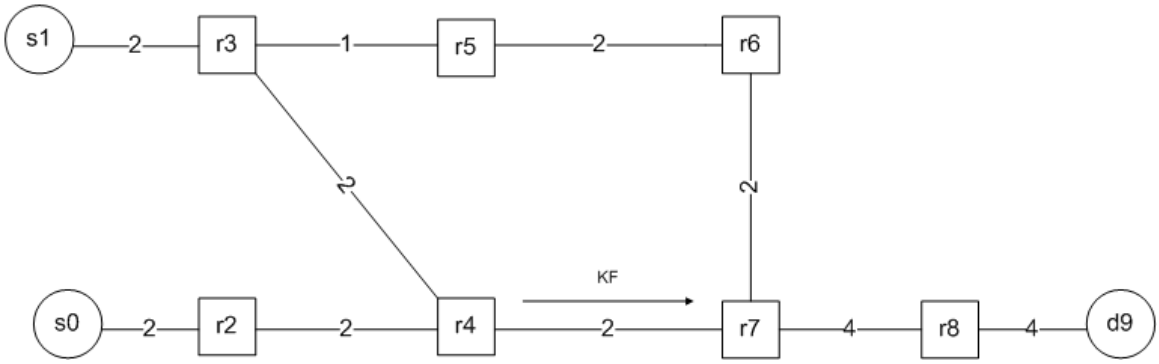

Figure 4.15: Network topology of DRQC flow reactivation simulation.

The network topology is the same as with the DRQC priority rerouting simulation. This topology gives network flows coming from s1 two viable paths to d9 and only one path to s0. A Kalman filter is placed on node r4's outbound queue to monitor the congestion that will happen at node r4. The network flows used in this simulation is listed in Table 4.6.

Table 4.6: The network flow table for the DRQC dynamically splitting flows simulation.

Flow	Priority	From	To	Bandwidth	Start Time	End Time
0	1	s0	d9	2	2	10
1	2	s1	d9	2	0	15

This simulation is similar to the DRQC priority rerouting simulation except that flow 1 is scheduled to run to 15 seconds instead of 10. The idea to this simulation is that when flow 0 stops streaming, the DRQC will be able to reactivate flow 1 because of the capacity that is now available. The results from this simulation are shown in Figure 4.16. This simulation was successful in demonstrating flow activation because at 11 seconds into the simulation, network flow 1 is reactivated after being paused at 3 seconds into the simulation. Network flow 1 was paused because of the network congestion detected in Figure 4.16 at 3 seconds and could not reroute because the link between node r3 and node r5 only has the capacity of 1 megabytes/second where network flow 1 requires a capacity of 2 megabytes/second.

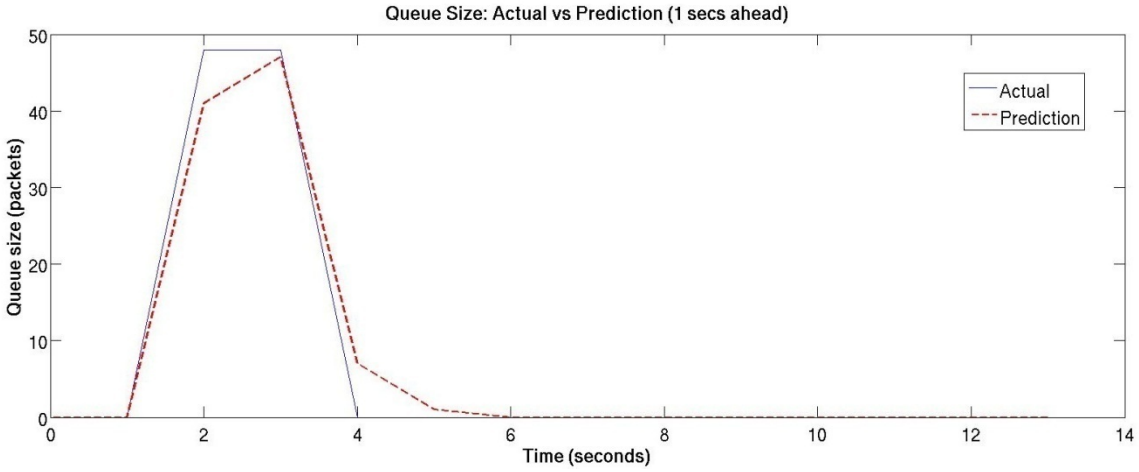

Figure 4.16: Graph of queue size in node 4 during the DRQC flow reactivation simulation.

DRQC Simulation: Comprehensive Case

The goal of this experiment is to demonstrate the DRQC's ability to utilize all of its features to optimize a mid size network. This simulation deals with a realistic network topology which is simulated for a longer time than the previous DRQC simulations. This allows a sufficient amount of time to evaluate the performance of the DRQC in a sensible network. This simulation creates a mid size network with multiple network flows. These network flows will cause congestion in the network. The DRQC will intervene with the network and fix the network congestion.

This experiment will have four different variations. The first variation will execute as normal without the use the implementation of the DRQC. This demonstrates how the network congestion will look like with no involvement from the DRQC. The network flows will cause congestion in the network. This will result in

full queues that will produce dropped packets. The queues will only return to normal when the network flows have finished streaming.

The second variation will utilize the DRQC decisions which are only based on the current network conditions data. The second variation will demonstrate the DRQC's ability to fix network congestion. This is shown by comparing the results from the first and second variation. The difference of network congestion will display the performance of the DRQC during the simulation. These comparisons will be done by analyzing the queue size graphs at the Kalman filter queues. Although the second variation only used the current queue size data and does not utilize the predicted queue data, the queue size graph can be used to look at the network congestion in that particular area of the network.

The third variation utilizes the DRQC which is based on both predicted and current network conditions data. This variation utilizes both predicted and current queue size data to make intelligent decisions to optimize the network. This variation will be used to demonstrate the DRQC's ability utilize predictions. This variation will be used to measure the performance of having access to predicted queue size data for the DRQC. By comparing the results from the second and third variation, this will show how the prediction data effects the reactions of the DRQC. This variation will be setup to make 1 second queue predictions.

The fourth variation will be the same as the third except it will utilize 5 second predictions. This will scale the performance of the DRQC when it has predictions further in the future. This variation will be compared with the third variation to see how the DRQC decisions are affected by further in the future predictions.

All the variations utilize the same network topology, network flows, and Kalman filter placements. The network topology has 14 nodes, 42 duplex links, and 6 Kalman filters which are illustrated in Figure 4.16. The network links all have the same capacity, which is 4 megabytes/second. The Kalman filters were placed in the middle of the network. Network flows were created by choosing a source node on one of the network and choosing a destination node on the other side of the network. This will create network congestion in the middle of the network, which is where the Kalman filter are located. Network flows were given a random priority between 1 and 5. Network flows were given a random constant sending rate between 1 megabytes/second and 4 megabytes megabytes/second. The network flows were given a random starting point and will be scheduled to run for 20 seconds. In this setup of a network sending rate and running time, a network flow will send between 1,000 and 4,000 packets a second. Each packet had the size of 1000 bytes. A network flow was set to start streaming at a random time between 1 and 100 seconds. This experiment is set to simulate this network for 120 seconds. Through this process of creating network flows, 31 network flows are created.

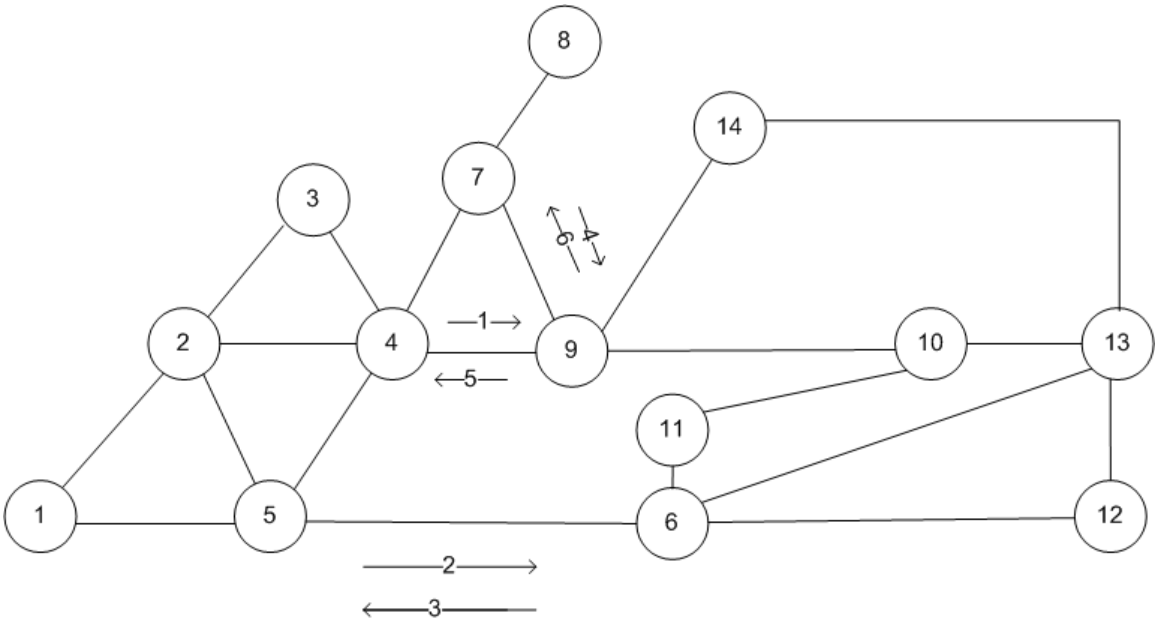

Figure 4.17: Network topology of the DRQC Simulation: Comprehensive Case Simulation. The arrows represent Kalman filters on the outbound queues.

The results from each Kalman filter are displayed in Figure 4.18, 4.19, 4.20, 4.21, 4.22, and 4.23. These figures show each individual Kalman filter's queue sizes during each variation of the DRQC comprehensive case simulations. The results are shown this way to make it easier to analysis the network congestion of this simulation by visualization. The larger the queue size, the larger the network congestion is in that queue.

The results from Kalman filter 1 are shown graphs in Figure 4.18. These graphs display the number of packets in the queue during each of the four variations mentioned earlier and how much network congestion is at this queue. In the top graph, Kalman filter 1 experiences congestion at two points and reaches maximum queue size

in the second spike of network congestion. When the DRQC is utilized without predictions, the second spike of network congestion is reduced so that the peak of congestion is at about 300 packets; this is displayed in the second graph in Figure 4.18. When the DRQC utilizes 1 second Kalman filter predictions, the second spike of congestion is reduced even further and has a peak of about 200 packets; this is displayed in the third graph in Figure 4.18. When the DRQC uses 5 second predictions, the network congestion is greater than using 1 second predictions. This was caused from inaccurate predictions as shown from the red dotted line in the fourth graph in Figure 4.18. Although the network congestion is greater, it is still less than when the DRQC was not used at all in the first graph in Figure 4.18.

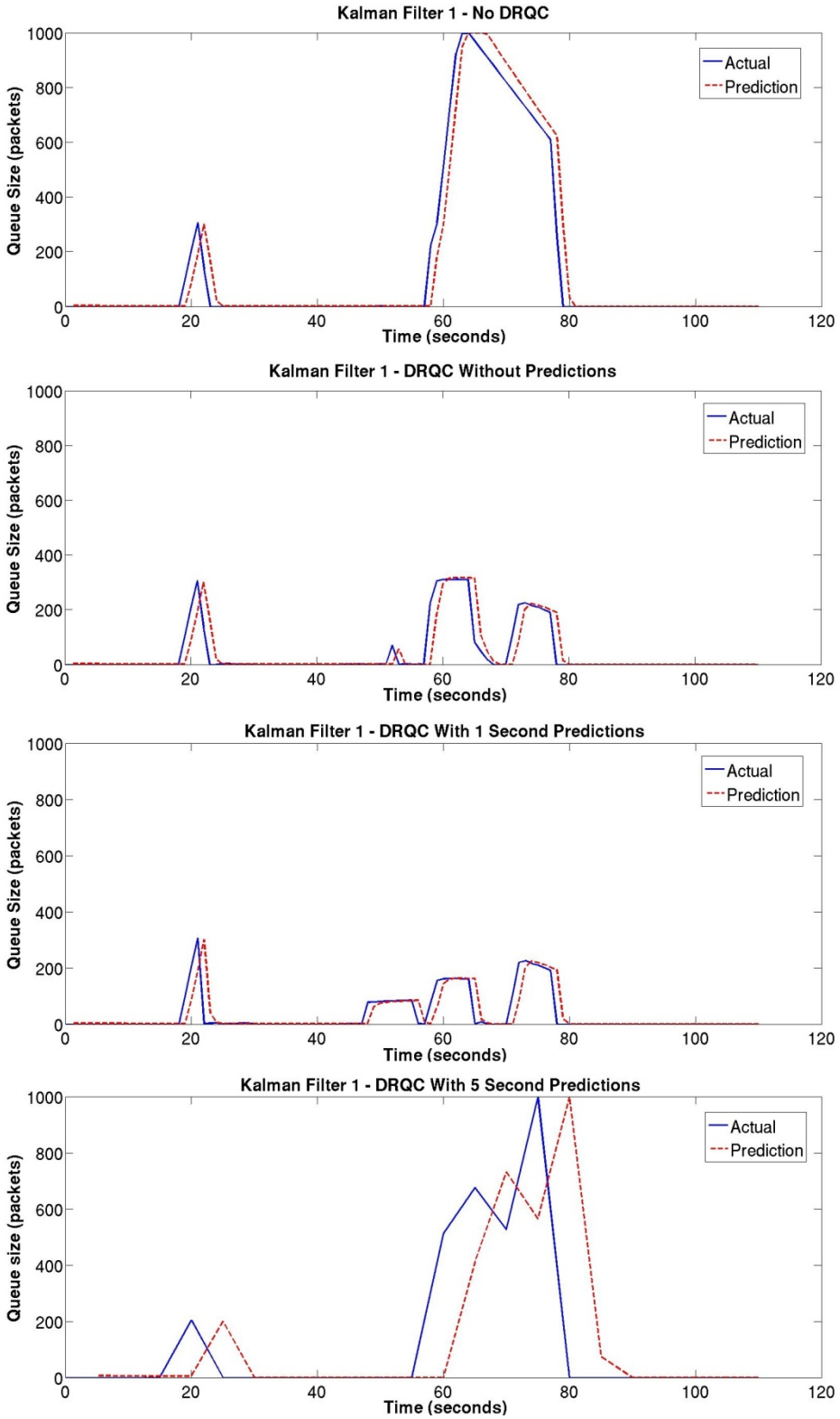

Figure 4.18: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 1.

The results from Kalman filter 2 are shown graphs in Figure 4.19. In the top graph, Kalman filter 2 experiences network congestion at two points and reaches maximum queue size in both spikes of network congestion. This is similar to Kalman filter 1 except that the first spike of network congestion did not reach maximum queue size of 1000 packets. When the DRQC is utilized without predictions, both spikes of network congestion is reduced so that the first spike has a peak of about 300 packets and the second spike is reduced to a shorter duration of maximum queue size. When the DRQC utilizes 1 second Kalman filter predictions, the first spike of network congestion is reduced again and one of the miniature spikes of congestion in the second spike is eliminated. The peak of the second spike of network congestion is reduced from 1000 to about 800 packets. When the DRQC uses 5 second predictions, the network congestion is greater than using 1 second predictions and without using predictions. Similar to Kalman filter 1, the accuracy of the predictions is causing a greater amount of network congestion. These results are still better than not using the DRQC at all, though.

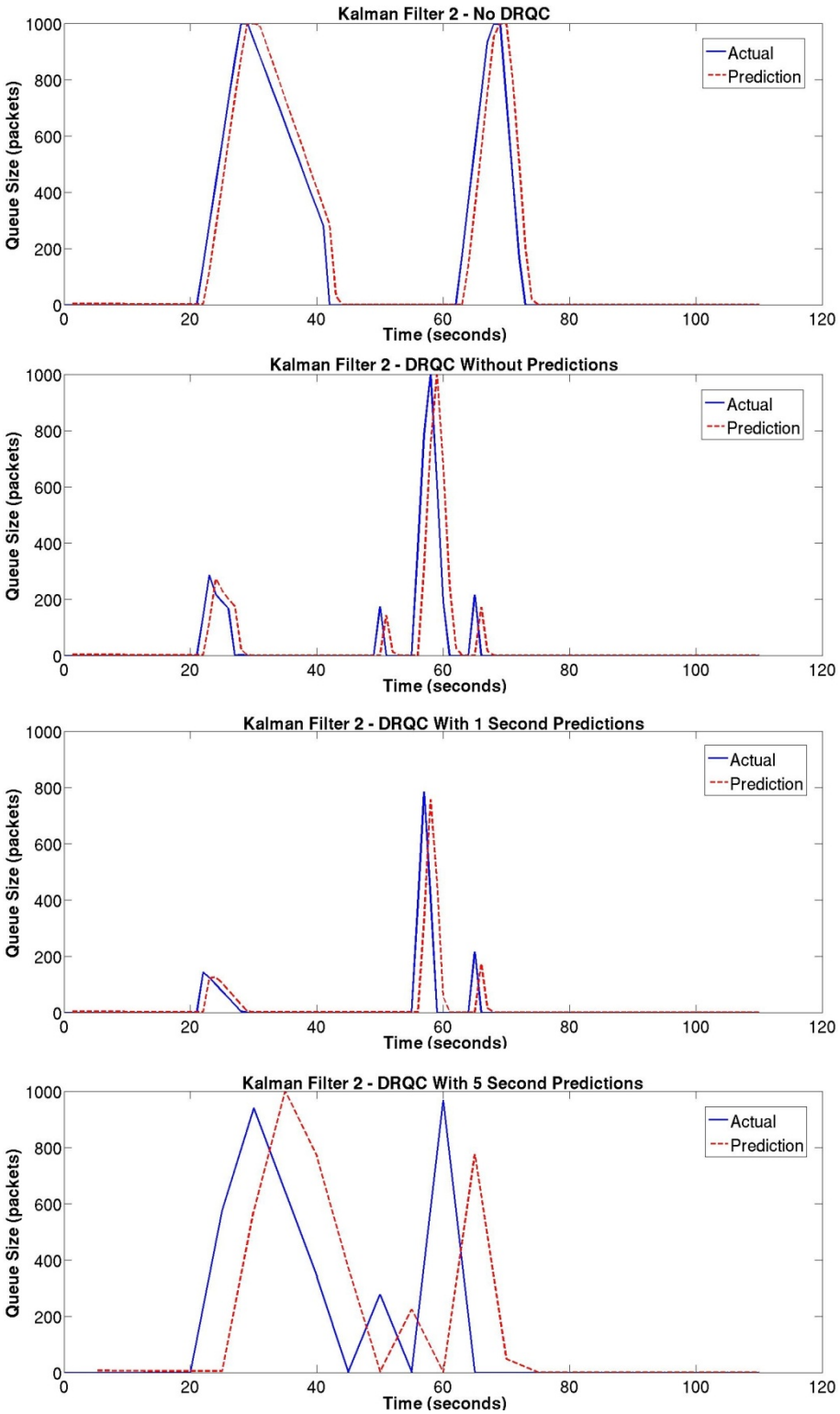

Figure 4.19: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 2.

The results from Kalman filter 3 are shown graphs in Figure 4.20. In the top graph, Kalman filter 3 experiences network congestion at one point and reaches maximum queue size during this point. When the DRQC is utilized without predictions, the network congestion at this point is reduced by almost a fourth. When the DRQC utilizes 1 second Kalman filter predictions, there is little change in network congestion but with the rate of reduced network congestion goes down compared to the previous variation of not using predictions. When the DRQC uses 5 second predictions, the network congestion is greater than using 1 second predictions and without using predictions. Similar to the previous Kalman filters, the accuracy of the predictions is causing a greater amount of network congestion. These results are still better than not using the DRQC at all, though.

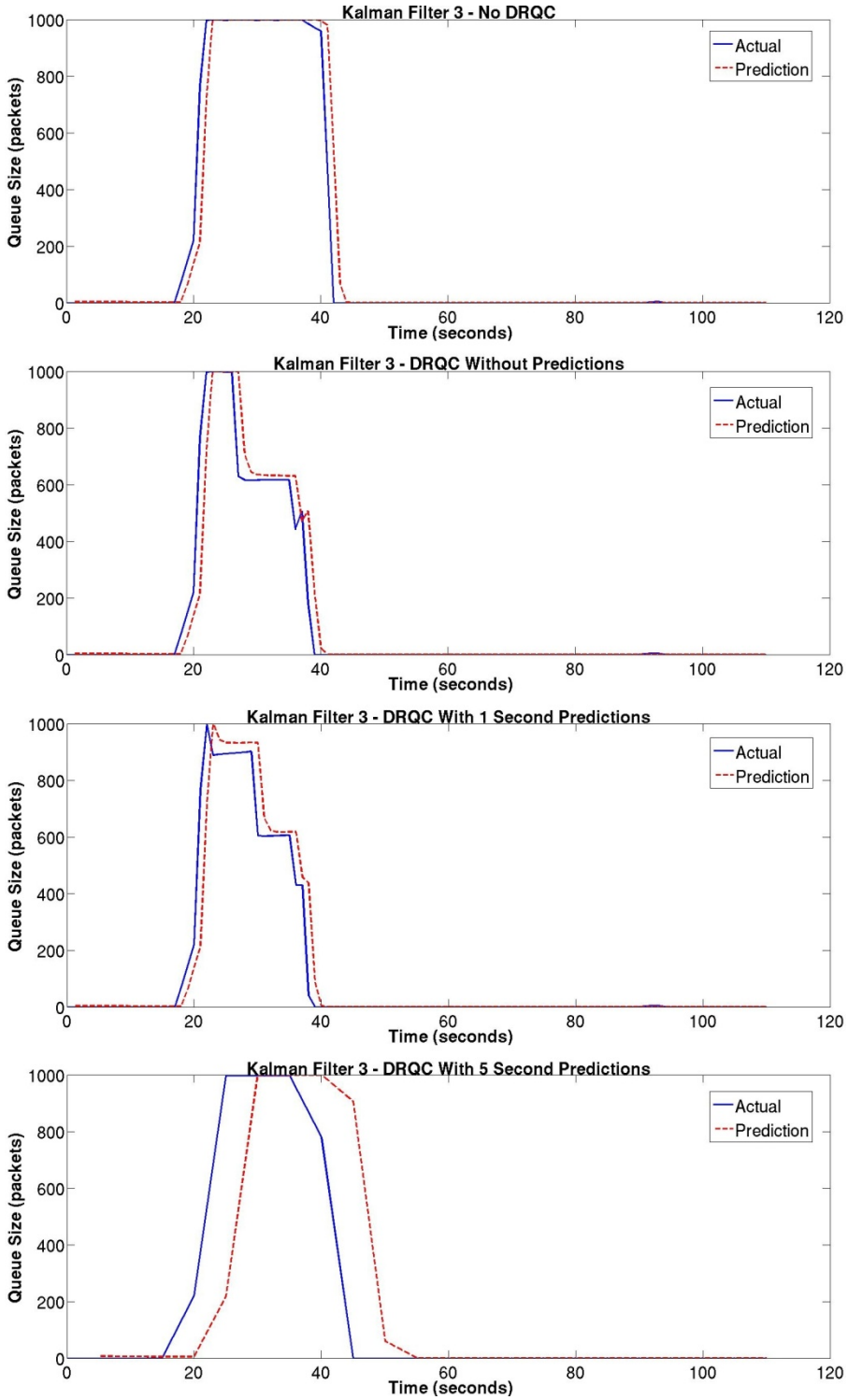

Figure 4.20: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 3.

The results from Kalman filter 4 are shown graphs in Figure 4.21. In the top graph, Kalman filter 4 experiences network congestion at two points and reaches maximum queue size during both of these points. When the DRQC is utilized without predictions, the network congestion at both of these points is reduced by more than half. When the DRQC utilizes 1 second Kalman filter predictions, the first spike of network congestion is reduced to a peak of 700 packets compared 950 packets when not utilizing predictions. When the DRQC uses 5 second predictions, the network congestion is again greater than using 1 second predictions and without using predictions. Similar to the previous Kalman filters, the accuracy of the predictions is causing a greater amount of network congestion. These results are still better than not using the DRQC at all, though.

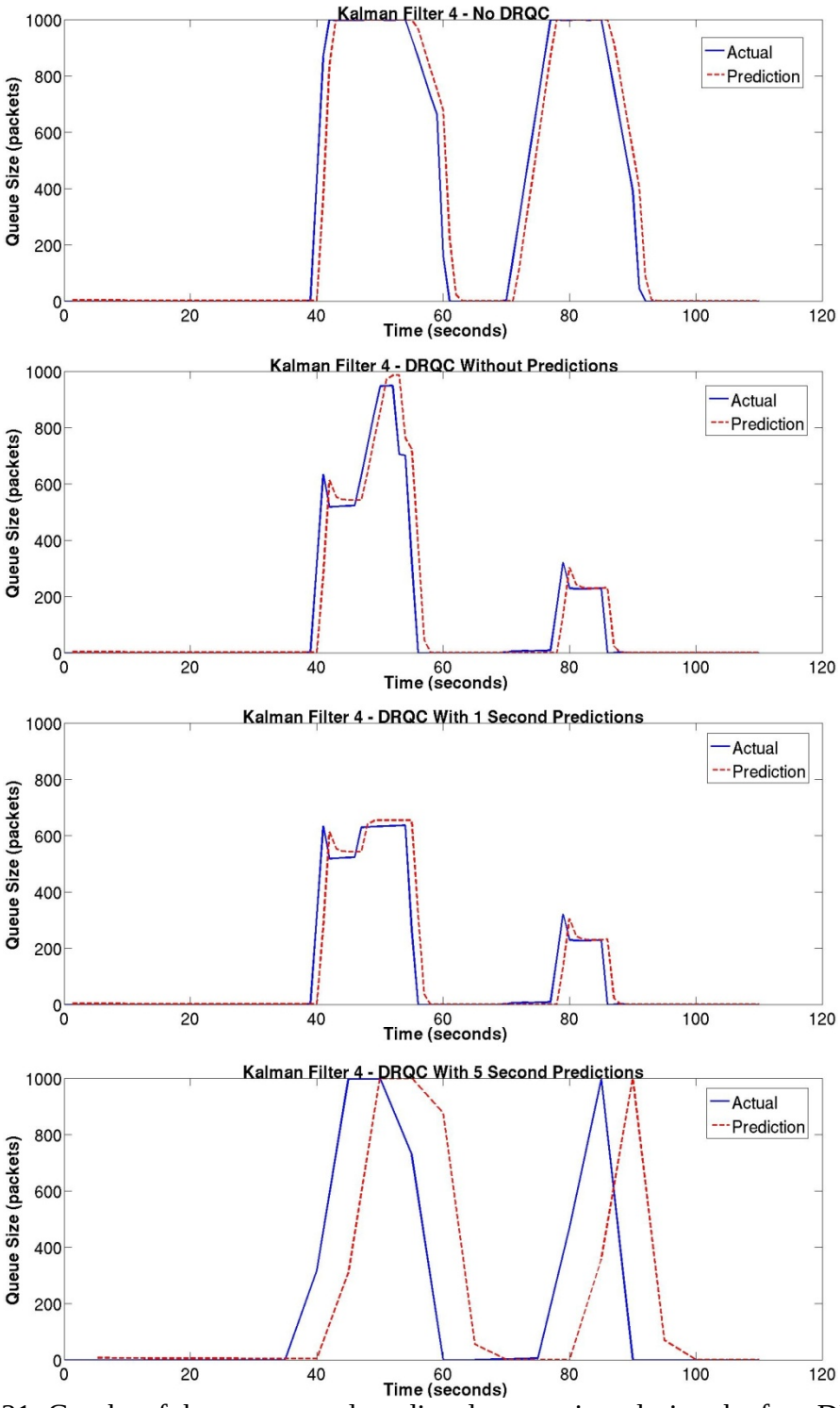

Figure 4.21: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 4.

The results from Kalman filter 5 are shown graphs in Figure 4.22. In the top graph, Kalman filter 3 experiences minor network congestion at one point during the simulation. When the DRQC is utilized without predictions, the duration of this network congest is reduced by almost 5 seconds and has a slightly less of a peak. When the DRQC utilizes 1 second Kalman filter predictions, the network congestion is the same as with not using predictions at all. The reason of this behavior is that when the queue size reaches enough network congestion where the DRQC rerouting process will execute, the network congestion is quickly fixed and the queue size returns to normal at a quick rate. When the DRQC uses 5 second predictions, the network congestion is again greater than using 1 second predictions and without using predictions. Similar to the previous Kalman filters, the accuracy of the predictions is causing a greater amount of network congestion. These results are still better than not using the DRQC at all, though.

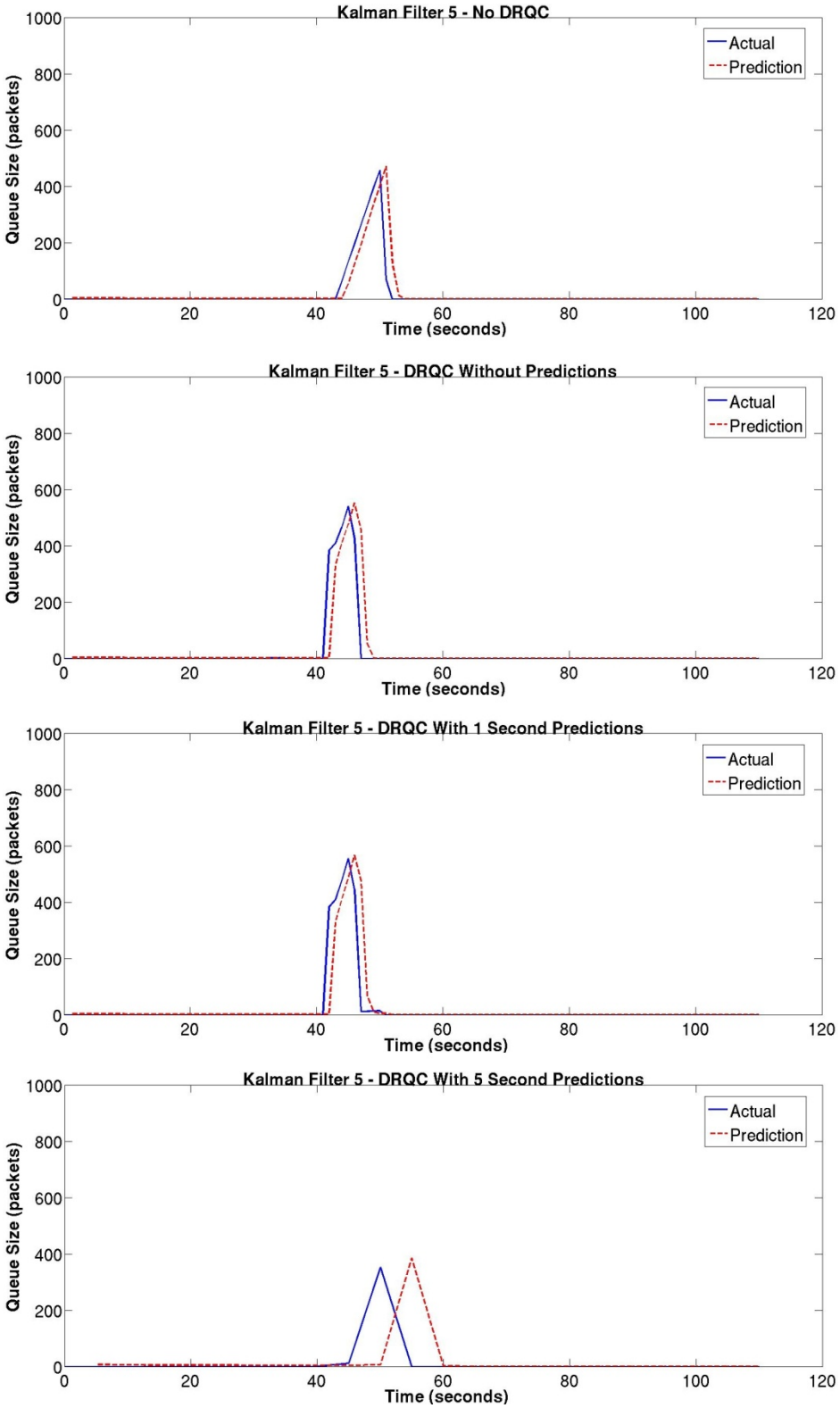

Figure 4.22: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 5.

The results from Kalman filter 6 are shown graphs in Figure 4.23. In the top graph, Kalman filter 6 experiences network congestion at one point and reaches maximum queue size. When the DRQC is utilized without predictions, the network congestion is reduced so that the queue size does not reach maximum. When the DRQC utilizes 1 second Kalman filter predictions, the first spike of network congestion is completely eliminated. The DRQC detected the future network congestion and avoided it through rerouting and pausing network flows. When the DRQC uses 5 second predictions, the network congestion is again greater than using 1 second predictions and without using predictions. Similar to the previous Kalman filters, the accuracy of the predictions is causing a greater amount of network congestion. These results are still better than not using the DRQC at all, though.

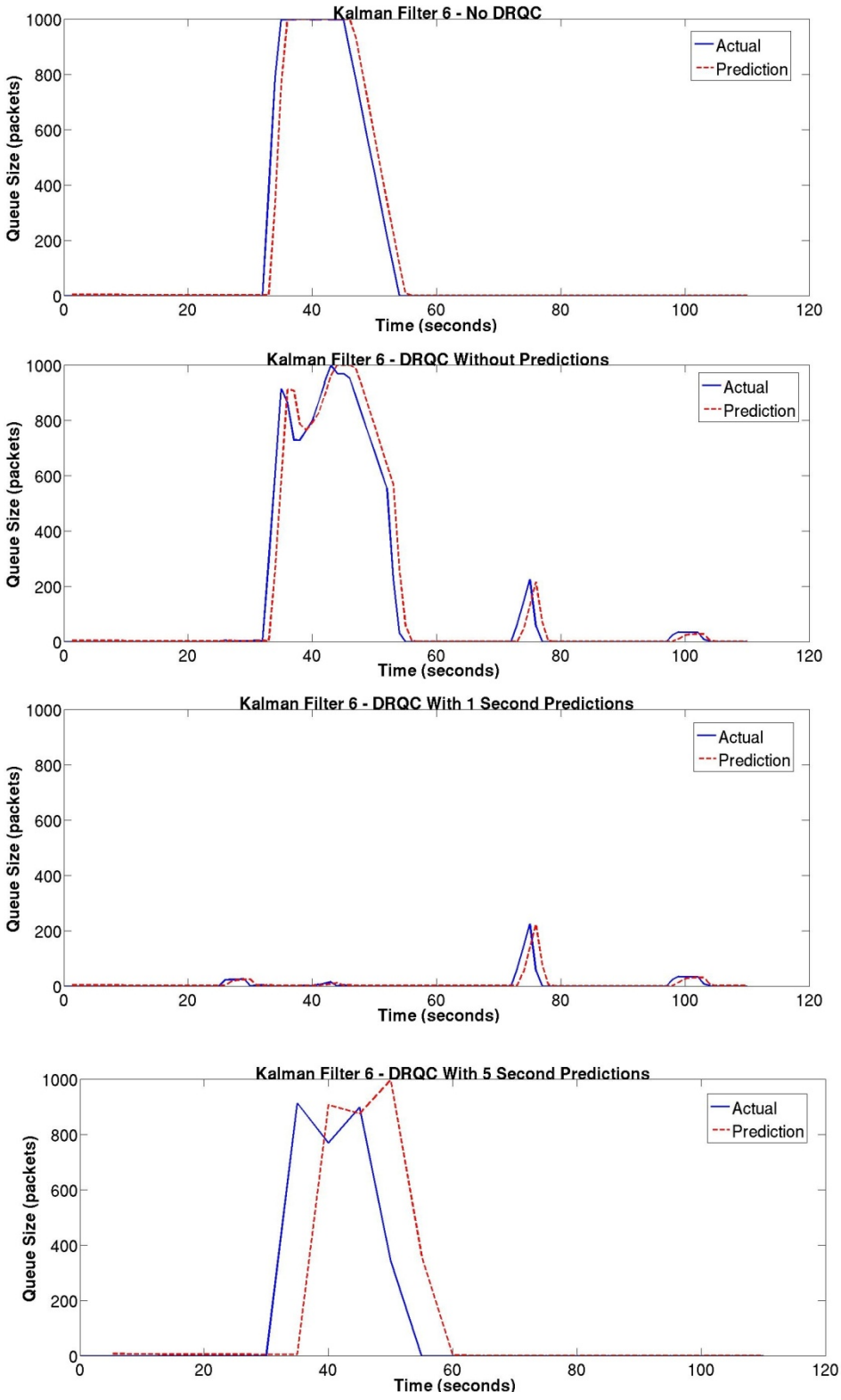

Figure 4.23: Graphs of the current and predicted queue sizes during the four DRQC comprehensive simulation variations at Kalman Filter 6.

The number of packets dropped during these simulations is shown in Figure 2.24. The number of packets dropped was recorded from the trace file produced at the end of the simulation. This number of dropped packets is calculated from each packet dropped in all the queues in the network during the simulation. The number of dropped packets weren't just from the six Kalman filters. In the first variation where the DRQC was not utilized, 27,512 packets were dropped. In the second variation where the DRQC was utilized but without prediction data, 1,383 packets were dropped. This showed that by utilizing the DRQC, the number of packets being dropped reduced greatly. In the third variation where the DRQC was utilized with 1 second prediction data, 366 packets were dropped. This demonstrates that the prediction data improved the DRQC's performance. In the fourth variation where the DRQC was utilized with 5 second prediction data, 5858 packets were dropped. This reveals that the predictions were inaccurate and that the DRQC actually starts performing worse than without using the predictions at all. Therefore, passing inaccurate predictions to the DRQC will make the network conditions more congested which results in more packets being dropped. The results from the fourth variation was better than the first variation, therefore using the DRQC with lesser accurate predictions was still was better than not using the DRQC at all.

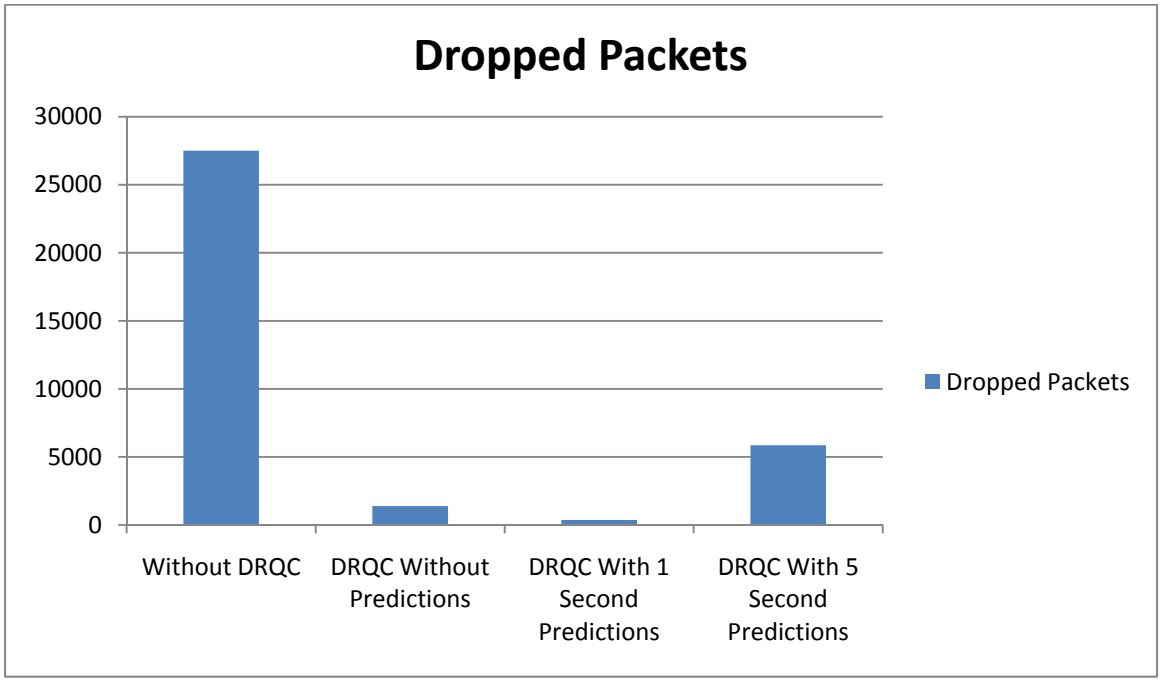

Figure 4.24: Chart of dropped packets in the DRQC Simulation: Comprehensive Case throughout the four variations.

All the simulations done with these variations took a significant amount of time to simulate and execute. In the first variation where the DRQC was not utilized, the simulation took 10 minutes to simulate. In that simulation, the Kalman filters were still predicting in order to record network congestion, so much of the processing time was from Kalman filter calculations made in MATLAB. In the second variation where the DRQC was utilized but without prediction data, the simulation took 33 minutes to simulate. In the third variation where the DRQC was utilized with 1 second prediction data, the simulation took 50 minutes to simulate. In the fourth variation where the DRQC was utilized with 5 second prediction data, the simulation took 48 minutes to simulate. All the simulations used the same machine to run. The operating system these simulations were executed on was CentOS release 5.5. The

machine had 2 processors, which both were Quad-Core AMD Opteron Model 2356 and has a core speed of 2300 MHz. The machine has 8 gigabytes of memory.

Analysis

Although previous research claimed to accurately predict queue sizes up to 100 seconds into the future, through experimentation of using these Kalman filters in more realistic networks, it was shown that Kalman filter predictions that far into the future are not accurate enough for a network controller to make decisions based on them. The reason why 100 second predictions were possible in past research is because the sending rate set very low, which was at 5 packets/sec [11]. That sending rate is not acceptable in most computer networks. By utilizing 1 second predictions, the Kalman filter predictions were shown that they can handle more realistic sending rates of between 1 megabyte/second and 4 megabytes/second. With packets being at the size of 1000 bytes, this results in between 1,000 and 4,000 packets being sent a second as opposed to 5 packets/second in previous research [11].

In order to make accurate Kalman filter predictions, there must be a measurement before a queue can fill up to maximum capacity. There is three ways of adapting a network environment to make accurate Kalman filter predictions. The first way is to adjust the sending rates of your network flows so that the queues will not fill up as fast. The second way is to increase your maximum queue size so it takes more packets to fill up the queue. Real router queue sizes are measured in the amount of memory and not in the number of packets [22]. One of the weaknesses in ns2 is that

the queue size is measured in the number of packets, no matter what size they are. So another way through ns2 is to increase the packet size, although this is not favorable for simulating a real router. The danger of increasing the packet size in a real network, though, is that the larger the packet size, the greater the risk of the packet being corrupted or dropped [3]. This is especially the case in wireless networks [3]. The third way to adapting a network is to set the Kalman filters to predict in less of a time frame. For example, if you wanted to double the sending rates of the last experiment and wanted to keep similar prediction accuracy, then you should make the Kalman filters predict at .5 seconds instead of 1 second. As mentioned before, another alternative method to creating similar Kalman filter results is where you can increase the maximum queue size to 2000 packets instead of 1000. This will compensate for the shorter amount of time a queue will reach maximum queue size and allow you to create an accurate Kalman filter.

The DRQC demonstrated that in a flooded network, it can reduce the network congestion dramatically. While not only reducing the network congestion, the DRQC considered which network flows were important by analyzing the priority. The high priority network flows were given optimal network capacities and allowed to stream with little or no interruption from network congestion. This algorithm would be valuable in network situations where network flows need a guaranteed QoS to achieve a mission.

Summary

In this chapter, Kalman filters were tested to see how they perform in a more realistic network environment. This validation process demonstrated the accuracy of the Kalman filter predictions in a mid size network. This information was required for the design and implementation of a network controller called the DRQC. With this information, the DRQC was implemented in ns2. The first set of DRQC simulations demonstrated the key features of the DRQC that were mentioned in Chapter III. In the DRQC Comprehensive Case Simulation, it demonstrated how the DRQC performed in a mid size network with multiple network flows. Without using prediction data, the DRQC reduced the network congest greatly. With accurate prediction queue size data, the DRQC reduced the network congestion. With prediction queue size data that predicted out to the future too far though, the DRQC performed worse than with using the prediction data at all but still better than without utilizing the DRQC.

V. Conclusions and Recommendations

Research Summary

While the current routing and congestion control algorithms in use today may be sufficient for networks with relatively static topology, theses algorithm may not be sufficient for military networks where a certain level of quality of service (QoS) needs to be achieved to complete a mission. Current networking technology limits a network's ability to adapt to changes and interactions in the network, often resulting in sub-optimal performance. The main cause of this situation is the lack of network context awareness available. Network research has shown that through the use of intelligent agents, it is possible to optimize a network. The main method of improving the network context awareness is by using queue size data. Through the utilization of predicted and current queue size data, this research demonstrates that it is possible to detect, locate, and fix network congestion.

This research proposes the use of predicted the queue sizes in a network through the utilization of Kalman filters. Past research has shown a Kalman filter can be utilized to make mid level prediction. This research demonstrated how Kalman filters perform in more realistic networks through simulation. With this information, a network controller was designed and implemented call the Dynamic Routing Queue Controller (DRQC). The DRQC can take in the current and predicted situation, and manage the network flows that are best fit for the mission. The network flows have a priority and the DRQC can ensure the highest priority flows reach their destination with little or no interference from network congestion.

Research Conclusions

This research presents a network controller that makes intelligent decisions to optimize a network. These intelligent decisions are based on Kalman filter predicted and current queue sizes. The DRQC demonstrated that given a flooded network where packets were being dropped because of network congestion, it could manage these network flows to fix the network congestion. With the use of accurate predicted queue sizes, it was possible for the DRQC to make decisions to prevent future network congestion. Even without predicted queue size data, the DRQC can still optimize a network and manage network flows to ensure the highest priority network flows.

Future Research Recommendations

This research points to a cornucopia of research areas in the field of network optimization. Further study of network context awareness and the utilization of intelligent agents implemented in a network environment are warranted to confirm the theory that an intelligent agent can optimize a network when it has enough information about the current and predicted network situation. This work proposes the use of Kalman filter predictions for a network controller to make decisions about the future state of a computer network.

Future work can be done with improving the prediction system or improving the Kalman filter algorithm. The modular design of the DRQC makes this possible and can easily be modified to use a different prediction system. Other queue size prediction

methods could be utilized by using a mean based system or any other prediction system. The DRQC provides a test bed for future queue size prediction systems. Adjustments can be made the any prediction system by measuring the performance of the DRQC.

The other category of extending this research work is improving the DRQC. These improvements would make the DRQC more realistic and robust. It would be valuable if the DRQC could handle node and link failures. Through this improvement, it would be possible for this controller to react to the network topology changes as ad hoc network tend to do often. Another improvement would be transforming this centralized system into a distributed system. There would be value in specifying a method of how the queue size data would be passed to a controller in a network. This is an issue that is necessary to address when taking this research from simulation to a real network.

There is also the possibility to improve the coding quality. The Kalman filters require the MATLAB 2007b libraries to operate. This code was directly from Mingook Kim's thesis work. If the code was converted to a lower language such as C++ or Java, than this could allow larger simulations to be ran in a feasible time period. When the C code has to call a MATLAB function, this slows down the running time of the simulation. If there are six Kalman filters that are predicting every 1 second for a simulation that takes 100 seconds, then the C code will have to make 600 MATLAB calls and slow down the processing time of the simulation.

Final Remarks

This research has shown by analyzing the queue sizes in a network, it is possible to detect, fix, and prevent network congestion. This radical approach can be used by intelligent agents to optimize a network. This research has shown there is promise of intelligent agents optimizing a network by analyzing the queue sizes in that network. It is possible to locate and find the source of network congestion in a given network. Through the utilization of Kalman filter predictions, the DRQC makes a significant contribution to network optimization and the use of network context awareness in a computer network. Improving network context awareness has the ability to allow network agent to optimize a network situation to achieve certain network wide goals. This research paves the way for future research of analyzing the queue size data to detect network congestion.

Appendix A: DRQC Class Diagram

At the beginning of the simulation, the original network situation is fed into the DRQC class which includes the links, nodes, flows, and other vital information that is needed to execute TCL commands to control the network. Figure A.1 contains the class diagram of the DRQC. The DRQC class is associated with the network through a modified version of the drop tail queue class in NS2. This Kalman filter drop tail class contains the code required to make Kalman filter predictions. All of the drop tail queues in the simulated network pass information about the next prediction to one instance of the DRQC class. The DRQC keeps track of the links and flows through an array of data structures. Through this method, DRQC contains the initial network and uses this initial network to push network flows through it during the rerouting process. So, at the beginning of the rerouting process, the DRQC uses the initial network and pushes the highest priority flow one at a time then updates these link capacities of this pseudo-network. These data structures contain vital information about the links and network flows that are used to make decisions on how to fix network congestion.

A major benefit of this design is the expandability and ability to modify the DRQC. For instance, if there is a different prediction system other than Kalman filter predictions, then the DRQC will be able to handle it. If there a different method of changing the routing tables needed, the DRQC can also adapt to this. The DRQC design is modular and only requires information about the network flows and the links.

-

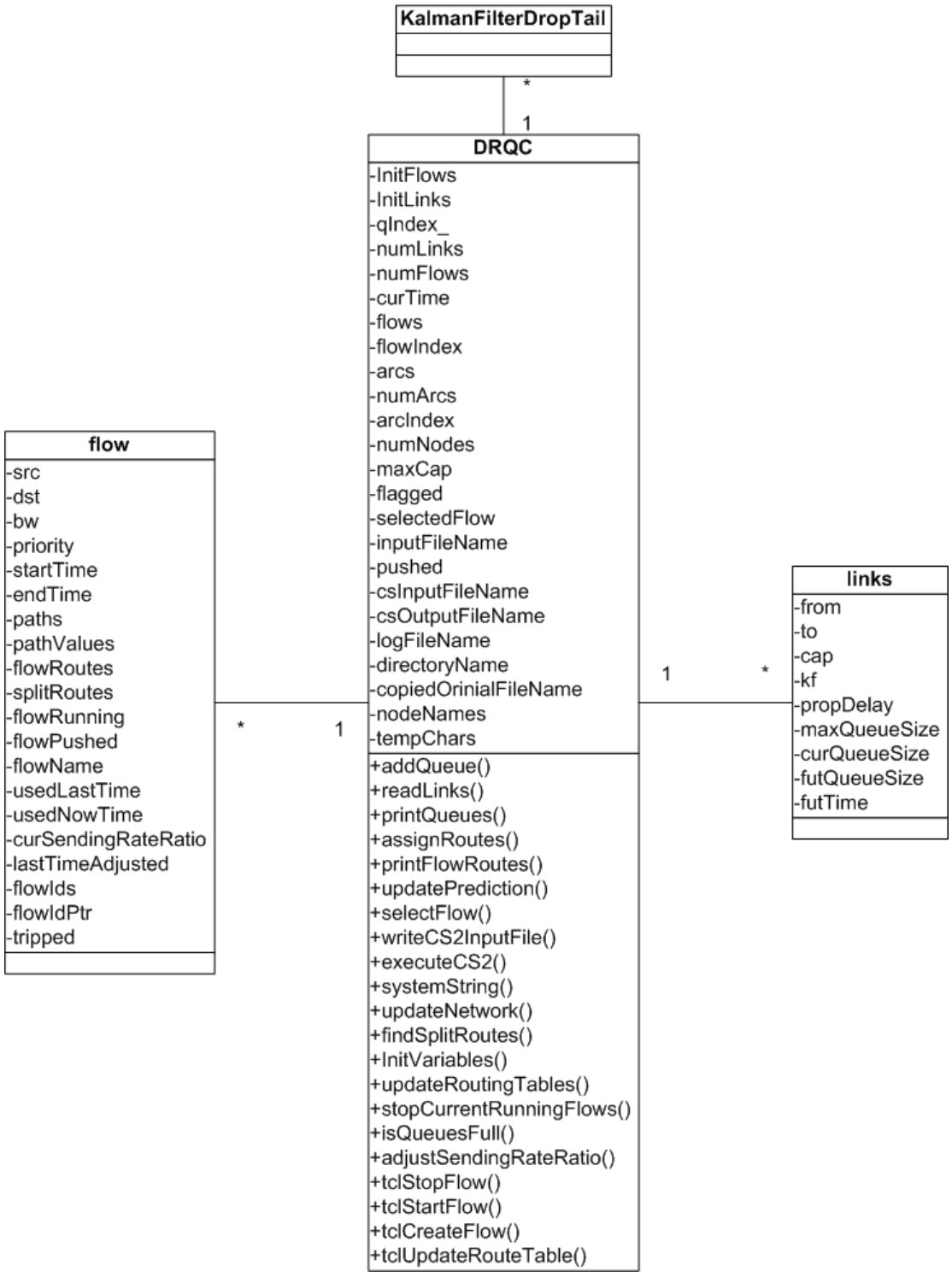

Figure A.1: Class diagram of DRQC.

Bibliography

[1] M. Compton, "The Network Tasking Order (NTO)," *IEEE Military Communications Conference*, pp. 1-7, 2008.

[2] J. M. Pecarina, "Creating an Agent Based Framework to Maximize Information Utility," Air Force Institue of Technology, WPAFB, OH, Thesis 2008.

[3] Nathan Stuckey, "Stochastic Estimation And Control Of Queues Within A Computer Network," Wright-Patterson AFB, Technical 2007.

[4] J. Vlach, "Network Theory and CAD," *Digital Object Identifer*, pp. 23-27, 1993.

[5] V. Swaminathan, "Network flow techniques for dynamic voltage scaling in hard real-time systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1385 - 1398, 2004.

[6] Jon Kleinberg, Eva Tardos, "Disjoint Paths in Directed and Undirected Graphs," in *Algorithm Design*.: Pearson Education, 2006, ch. 7, pp. 376-379.

[7] Jon Kleinberg, Eva Tardos, *Algorithm Desgin*., 2005, p. 340.

[8] Dan Simon. (2009, April) Innovatia. [Online]. http://www.innovatia.com/software/papers/kalman.htm

[9] Mingook Kim, "Stochastic Estimation and Control of Queues Within a Computer Network," Wright-Patterson AFB, 2009.

[10] Yuanqing Xia; Jizong Shang; Jie Chen; Guo-Ping Liu;, "Networked Data Fusion With Packet Losses and Variable Delays," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, pp. 1107 - 1120 , 2009.

[11] Mingook Kim, "Stochastic Estimation and Control Of Queues within a Computer Network," Air Force Institute of Technology, WPAFB, OH, Thesis 2009.

[12] Peter S. Maybeck, "The Kalman Filter: Introduction to Concepts," in *Stochastic models, estimation, and control*. New York, NY, OH: Academic Press, 1979, ch.

1, pp. 3-15.

[13] Greg Welch, Gary Bishop, "An Introduction to the Kalman Filter," *Annual Conference on Computer Graphics & Interactive Techniques*, 2006.

[14] Song Ci, Sharif H,Young A, "A link adaptation approach for QoS enhancement in wireless networks," *Local Computer Networks*, pp. 373 - 374, 2001.

[15] Periklis Tsingotjidis, Jeremiah F. Hayes, Hyong S. Kim, "Esimation and Prediction Approach to Congestion Control in ATM Newtworks," *Global Telecommunications Conference*, pp. 1785 - 1789, 1994.

[16] Rich Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," *High Performance Distributed Computing*, pp. 316 - 325, 1997.

[17] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, "Application level scheduling on distributed heterogeneous networks," *Proceedings of Supercomputing*, 1996.

[18] Matthew Allen, Rich Wolski, James Plank, "Adaptive Timeout Discovery using the Network Weather Service," *High Performance Distrubed Computing*, pp. 35-41, 2002.

[19] Teerawat Issariyakul, Ekram Hossain, *Introduction to Network Simulator NS2*. New York, USA: Springer, 2010.

[20] A. V. Goldberg, *J. Algorithms*, pp. 1-29, 1997.

[21] A. Penttinen, Chapter 8 - Queueing Systems.

[22] (1999) Cisco Systems. [Online]. http://www.cisco.com/networkers/nw99_pres/601.pdf

[23] M. Gocmen, "The Benefits of a Network Tasking Order in Combat Search and Rescue Missions," *IEEE Military Communications Conference (MILCOM)*, pp. 1-7, 2009.

[24] A. Tiwari, "Is Communication Planning Using Mission Information Logistically Feasible?," *IEEE Military Communications Conference (MILCOM)*, pp. 1-7, 2009.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
24-03-2011	Master's Thesis	June 2009 – March 20011

4. TITLE AND SUBTITLE

ADAPTIVE QUALITY OF SERVICE ENGINE WITH DYNAMIC QUEUE CONTROL

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

James Haught

5d. PROJECT NUMBER
ENG 11G222

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/11-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Robert Bonneau
Program Manager
Air Force Office of Scientific Research (AFOSR/NL)
Software and Systems
Telephone number: (703) 696-9545
E-mail address: Robert.bonneau@afosr.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S) AFOSR/NL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

While the current routing and congestion control algorithms in use today are often sufficient for networks with relatively static topology, these algorithms may not be sufficient for military networks where a certain level of quality of service (QoS) needs to be achieved to complete a mission. Current networking technology limits a network's ability to adapt to changes and interactions in the network, often resulting in sub-optimal performance. This research investigates the use of queue size predictions to create a network controller to optimize computer networks. These queue size predictions are made possible through the use of Kalman filters to detect network congestion. The premise is that intelligent agents can use such predictions to form context-aware, cognitive processes for managing communication in mobile networks. The network controller designed and implement in this thesis will take in the current and predicted network conditions and make intelligent choices to optimize the network.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr Kenneth Hopkinson (ENG)
U	**U**	**U**	**UU**	107	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4579 (Kenneth.Hopkinson@afit.edu)